异步图书 www.epubit.com

用Python实现数据操作和复杂数据分析

Packt

Python数据分析

（第2版）

Python Data Analysis

Second Edition

[美] 阿曼多·凡丹戈（Armando Fandango）著

韩波 译

中国工信出版集团

人民邮电出版社
POSTS & TELECOM PRESS

Python数据分析

（第2版）

Python Data Analysis

Second Edition

[美] 阿曼多·凡丹戈（Armando Fandango）著

韩波 译

人 民 邮 电 出 版 社

北 京

图书在版编目（ＣＩＰ）数据

Python数据分析 / (美) 阿曼多·凡丹戈
(Armando Fandango) 著 ; 韩波译. -- 2版. -- 北京 :
人民邮电出版社, 2018.6
ISBN 978-7-115-48117-7

Ⅰ. ①P… Ⅱ. ①阿… ②韩… Ⅲ. ①软件工具—程序
设计 Ⅳ. ①TP311.561

中国版本图书馆CIP数据核字(2018)第052547号

版 权 声 明

◆ 著　[美] 阿曼多·凡丹戈（Armando Fandango）
译　韩　波
责任编辑　胡俊英
责任印制　焦志炜

◆ 人民邮电出版社出版发行　北京市丰台区成寿寺路 11 号
邮编　100164　电子邮件　315@ptpress.com.cn
网址　http://www.ptpress.com.cn
固安县铭成印刷有限公司印刷

◆ 开本：800×1000　1/16
印张：18.25
字数：362 千字　2018 年 6 月第 2 版
印数：12 001 – 14 400 册　2018 年 6 月河北第 1 次印刷

著作权合同登记号　图字：01-2017-8632 号

定价：69.00 元

读者服务热线：(010) 81055410　印装质量热线：(010) 81055316
反盗版热线：(010) 81055315

内容提要

Python 作为一种高级程序设计语言，凭借其简洁、易读及可扩展性日渐成为程序设计领域备受推崇的语言。同时，Python 语言的数据分析功能也逐渐为大众所认可。

本书就是一本介绍如何用 Python 进行数据分析的学习指南。全书共 12 章，从 Python 程序库入门、NumPy 数组和 Pandas 入门开始，陆续介绍了数据的检索、数据加工与存储、数据可视化等内容。同时，本书还介绍了信号处理与时间序列、应用数据库、分析文本数据与社交媒体、预测性分析与机器学习、Python 生态系统的外部环境和云计算、性能优化及分析、并发性等内容。在本书的最后，还采用 3 个附录的形式为读者补充了一些重要概念、常用函数以及在线资源等重要内容。

本书延续了上一版示例丰富、简单易懂的优点，非常适合对 Python 语言感兴趣或者想要使用 Python 语言进行数据分析的读者参考阅读。

作者简介

Armando Fandango 是 Epic 工程咨询集团首席数据科学家，负责与国防和政府机构有关的保密项目。Armando 是一位技术精湛的技术人员，拥有全球创业公司和大型公司的工作经历和高级管理经验。他的工作涉及金融科技、证券交易所、银行、生物信息学、基因组学、广告技术、基础设施、交通运输、能源、人力资源和娱乐等多个领域。

Armando 在预测分析、数据科学、机器学习、大数据、产品工程、高性能计算和云基础设施等项目中工作了十多年。他的研究兴趣横跨机器学习、深度学习和科学计算等领域。

我要特别感谢我的妻子在编写本书过程中给予我的支持。同时，我还要感谢 UCF 的 Paul Wiegand 博士，他总是鼓励我寻求机会。最后，我非常感谢 Packt 团队的 Tushar、Sumeet、Amrita、Deepti 以及其他工作人员，感谢他们为本书的面世所付出的巨大努力。

技术审稿人简介

Joran Beasley 毕业于爱达荷大学，获得了计算机科学学士学位。7 年以来，他一直从事基于 Python 的桌面应用程序编程，同时将其应用于监控农业的大规模传感器网络。目前，他居住于爱达荷州莫斯科市，作为软件工程师供职于 METER 集团。

我要感谢我的妻子 Nicole，当我把大量的时间倾注在写作上的时候，她在默默地抚养我们的两个孩子。

Ratan Kumar 在过去的几年里一直在利用各种语言和技术从事软件编程。2013 年以来，Ratan Kumar 在 Web 服务领域使用 Python 语言，他发现无论是在个人项目还是专业项目方面，Python 都是最优雅、最高效、最易于接受的编程语言之一。Ratan 目前住在班加罗尔，是一个旨在简化股票市场投资的小型团队的核心成员。

前言

数据分析在自然科学、生物医学和社会科学领域有着悠久的历史。随着数据科学的发展，数据分析也呈现流行之势，几乎已经渗透到工业的方方面面。与数据科学类似，数据分析也致力于从数据中提取有效信息。为此，我们需要用到统计学、机器学习、信号处理、自然语言处理和计算机科学领域中的各种技术。

在第 1 章中，我们将给出一幅描绘与数据分析相关的 Python 软件的脑图。首先要知道的是，Python 生态系统已经非常完备，具有诸如 NumPy、SciPy 和 Matplotlib 等著名的程序包。当然，这没有什么好奇怪的，因为 Python 在 1989 年就诞生了。Python 易学、易用，而且与其他程序设计语言相比语法简练，可读性非常强，即使从未接触过 Python 的人，也可以在几天内掌握该语言的基本用法，对熟悉其他编程语言的人来说尤其如此。你无需太多的基础知识，就能顺畅地阅读本书。此外，关于 Python 的书籍、课程和在线教程也非常多。

本书内容

第 1 章“Python 程序库入门”手把手地指导读者正确安装配置 Python 和基础的 Python 数值分析软件库。同时，本章还会展示如何通过 NumPy 创建一个小程序以及如何利用 Matplotlib 来绘制简单的图形。

第 2 章“NumPy 数组”介绍 NumPy 和数组的基础知识。通过阅读本章，读者能够基本掌握 NumPy 数组及其相关函数。

第 3 章“Pandas 入门”阐述 Pandas 的基本功能，其中涉及 Pandas 的数据结构与相应的操作。

第 4 章“统计学与线性代数”对线性代数和统计函数做了简要回顾。

第 5 章“数据的检索、加工与存储”介绍如何获取不同格式的数据，以及原始数据的清洗和存储方法。

第 6 章“数据可视化”介绍如何利用 Matplotlib 和 Pandas 的绘图函数来实现数据的可视化。

第 7 章“信号处理与时间序列”利用太阳黑子周期数据来实例讲解时间序列和信号处理，同时还会介绍一些相关的统计模型。本章使用的主要工具是 NumPy 和 SciPy。

第 8 章“应用数据库”介绍各种数据库和有关 API 的知识，其中包括关系数据库和 NoSQL 数据库。

第 9 章“分析文本数据和社交媒体”考察基于文本数据的情感分析和主题抽取。同时，本章还将为读者展示一个网络分析方面的实例。

第 10 章“预测性分析与机器学习”通过一个例子来说明人工智能在天气预报上的应用，这主要借助于 scikit-learn。不过，有些机器学习算法在 scikit-learn 中尚未实现，所以有时还要求助其他 API。

第 11 章“Python 生态系统的外部环境和云计算”将提供各种实例，来说明如何集成非 Python 编写的现有代码。此外，本章还将为读者演示如何在云中使用 Python。

第 12 章“性能优化、性能分析与并发性”为读者介绍通过性能分析（Profling）和 Cython 等关键技术来改善性能的各种技巧，同时还为读者介绍多核和分布式系统方面的相关框架。

附录 A“重要概念”将对本书中涉及的重要概念进行简要介绍。

附录 B“常用函数”概述本书中用到的程序库中的各种函数，以便于读者查阅。

读者须知

本书中的示例代码可以在大部分现代操作系统上运行；所有章节中的代码都需要用到 3.5.0 版本以上的 Python 和 pip3 软件。Python 3.5.x 的下载地址为 https://www.python.org/downloads/，这个页面上不仅提供了用于 Windows 和 Mac OS X 的安装程序，也提供了用于 Linux、UNIX 和 Mac OS X 系统的 Python 源代码。多数情况下，我们都得通过运行以下命令来以管理员权限安装各种本书要用到的 Python 库：

```
$ pip3 install <some library>
```

以下是用于本书中的示例的 Python 库的清单。

- numpy
- scipy
- Pandas
- Matplotlib
- ipython
- jupyter
- notebook
- readline
- scikit-learn
- rpy2
- Quandl
- statsmodels
- feedparser
- beautifulsoup4
- lxml
- numexpr
- tables
- openpyxl
- xlsxwriter
- xlrd
- pony
- dataset
- pymongo
- redis
- python3-memcache
- cassandra-driver
- sqlalchemy
- nltk
- networkx
- theanets
- nose_parameterized
- pydot2
- deap
- JPype1
- gprof2dot
- line_profiler
- cython
- cytoolz
- joblib
- bottleneck
- jug
- mpi4py

除了各种 Python 库之外，我们还需要以下软件。

- Redis server
- Cassandra
- Java 8
- Graphviz
- Octave
- R
- SWIG
- PCRE
- Boost
- gfortran
- MPI

通常情况下，对于以上的程序化和软件，我们应该选用最新版本。

> 以上列出的某些软件只是用于某个示例，因此在安装之前，请先检查该软件是否仅限用于某个示例代码。

对于通过 pip 安装的 Python 程序包，卸载方法如下所示：

```
$ pip3 uninstall <some library>
```

目标读者

本书的目标读者是对 Python 和数学有基本了解，并且想进一步学习如何利用 Python 软件进行数据分析的朋友。我们力争让本书简单易懂，但无法保证所有主题都面面俱到。如果需要，读者可以经由 Khan Academy、Coursera 之类的在线资源来复习自己的数学知识。

下列 Packt 出版社的书籍是推荐给读者的进阶读物。

- Building Machine Learning Systems with Python, Willi Richert and Luis Pedro Coelho (2013)
- Learning Cython Programming, Philip Herron (2013)
- Learning NumPy Array, Ivan Idris (2014)
- Learning scikit-learn: Machine Learning in Python, Raúl Garreta and Guillermo Moncecchi (2013)
- Learning SciPy for Numerical and Scientific Computing, Francisco J. Blanco-Silva (2013)
- Matplotlib for Python Developers, Sandro Tosi (2009)
- NumPy Beginner's Guide - Second Edition, Ivan Idris (2013)
- NumPy Cookbook, Ivan Idris (2012)
- Parallel Programming with Python, Jan Palach (2014)
- Python Data Visualization Cookbook, Igor Milovanović (2013)
- Python for Finance, Yuxing Yan (2014)
- Python Text Processing with NLTK 2.0 Cookbook, Jacob Perkins (2010)

排版约定

本书中，不同类型的信息会采用不同的排版样式，以示区别。下面针对各种排版样式及其含义进行举例说明。

文本、数据库表名、文件夹名、文件名、文件扩展名和路径名、伪 URL、用户输入和推特句柄（Twitter handles）中出现的代码文字，会显示：“如果您的当前登录账户没有足

够权限的话，则需要在这条命令前面加上 sudo。”

代码段显示格式如下。

```
def pythonsum(n):
   a = list(range(n))
   b = list(range(n))
   c = []

   for i in range(len(a)):
       a[i] = i ** 2
       b[i] = i ** 3
       c.append(a[i] + b[i])

   return c
```

所有的命令行输入或者输出内容会显示为如下格式。

```
$ pip3 install numpy scipy pandas matplotlib jupyter notebook
```

警告或者重要的提示在此显示。

小技巧在此显示。

资源与支持

本书由异步社区出品，社区（https://www.epubit.com/）为您提供相关资源和后续服务。

配套资源

本书提供如下资源：

- 本书源代码；
- 书中彩图文件。

要获得以上配套资源，请在异步社区本书页面中点击 配套资源 ，跳转到下载界面，按提示进行操作即可。注意：为保证购书读者的权益，该操作会给出相关提示，要求输入提取码进行验证。

如果您是教师，希望获得教学配套资源，请在社区本书页面中直接联系本书的责任编辑。

提交勘误

作者和编辑尽最大努力来确保书中内容的准确性，但难免会存在疏漏。欢迎您将发现的问题反馈给我们，帮助我们提升图书的质量。

当您发现错误时，请登录异步社区，按书名搜索，进入本书页面，点击“提交勘误”，输入勘误信息，单击“提交”按钮即可。本书的作者和编辑会对您提交的勘误进行审核，确认并接受后，您将获赠异步社区的 100 积分。积分可用于在异步社区兑换优惠券、样书或奖品。



与我们联系

我们的联系邮箱是 contact@epubit.com.cn。

如果您对本书有任何疑问或建议，请您发邮件给我们，并请在邮件标题中注明本书书名，以便我们更高效地做出反馈。

如果您有兴趣出版图书、录制教学视频，或者参与图书翻译、技术审校等工作，可以发邮件给我们；有意出版图书的作者也可以到异步社区在线提交投稿（直接访问www.epubit.com/selfpublish/submission 即可）。

如果您是学校、培训机构或企业，想批量购买本书或异步社区出版的其他图书，也可以发邮件给我们。

如果您在网上发现有针对异步社区出品图书的各种形式的盗版行为，包括对图书全部或部分内容的非授权传播，请您将怀疑有侵权行为的链接发邮件给我们。您的这一举动是对作者权益的保护，也是我们持续为您提供有价值的内容的动力之源。

关于异步社区和异步图书

“异步社区”是人民邮电出版社旗下 IT 专业图书社区，致力于出版精品 IT 技术图书和相关学习产品，为作译者提供优质出版服务。异步社区创办于 2015 年 8 月，提供大量精品 IT 技术图书和电子书，以及高品质技术文章和视频课程。更多详情请访问异步社区官网 https://www.epubit.com。

“异步图书”是由异步社区编辑团队策划出版的精品 IT 专业图书的品牌，依托于人民邮电出版社近 30 年的计算机图书出版积累和专业编辑团队，相关图书在封面上印有异步图书的 LOGO。异步图书的出版领域包括软件开发、大数据、AI、测试、前端、网络技术等。

异步社区

微信服务号

目录

第 1 章
Python 程序库入门

欢迎来到 Python 数据分析的世界！如今，Python 已成为数据分析和数据科学事实上的标准语言和标准平台之一。我们将为读者展示一张思维导图，图 1-1 中将给出 Python 生态系统为数据分析师和数据科学家提供的各种程序库。NumPy、SciPy、Pandas 和 Matplotlib 库共同构成了 Python 数据分析的基础，当前它们已经成为 SciPy Stack 1.0 的组成部分。在本章中，我们不仅会学习如何安装 SciPy Stack 1.0 和 Jupyter Notebook，还将编写一些简单的数据分析代码，为后面的学习做好热身。

下面是 Python 生态系统为数据分析师和数据科学家提供的常用程序库。

- **NumPy**：这是一个通用程序库，不仅支持常用的数值数组，同时提供了用于高效处理这些数组的函数。
- **SciPy**：这是 Python 的科学计算库，对 NumPy 的功能进行了大量扩充，同时也有部分功能是重合的。Numpy 和 SciPy 曾经共享基础代码，后来分道扬镳了。
- **Pandas**：这是一个用于数据处理的程序库，不仅提供了丰富的数据结构，同时为处理数据表和时间序列提供了相应的函数。
- **Matplotlib**：这是一个 2D 绘图库，在绘制图形和图像方面提供了良好的支持。当前，Matplotlib 已经并入 SciPy 中并支持 NumPy。
- **IPython**：这个库为 Python 提供了强大的交互式 Shell，也为 Jupyter 提供了内核，同时还支持交互式数据可视化功能。我们将在本章稍后介绍 IPython shell。
- **Jupyter Notebook**：它提供了一个基于 Web 的交互式 shell，可以创建和共享支持可实时代码和可视化的文档。Jupyter Notebook 通过 IPython 提供的内核支持多个版本的 Python。本章稍后将会为读者进一步介绍 Jupyter Notebook。

对于本书而言，当需要安装软件时，我们会在恰当的时机给出相应的安装说明。在安装软件的过程中遇到困难或者不能断定最佳方案时，读者可以参考图 1-1，这里提供了寻找解决问题所需辅助信息的指南。

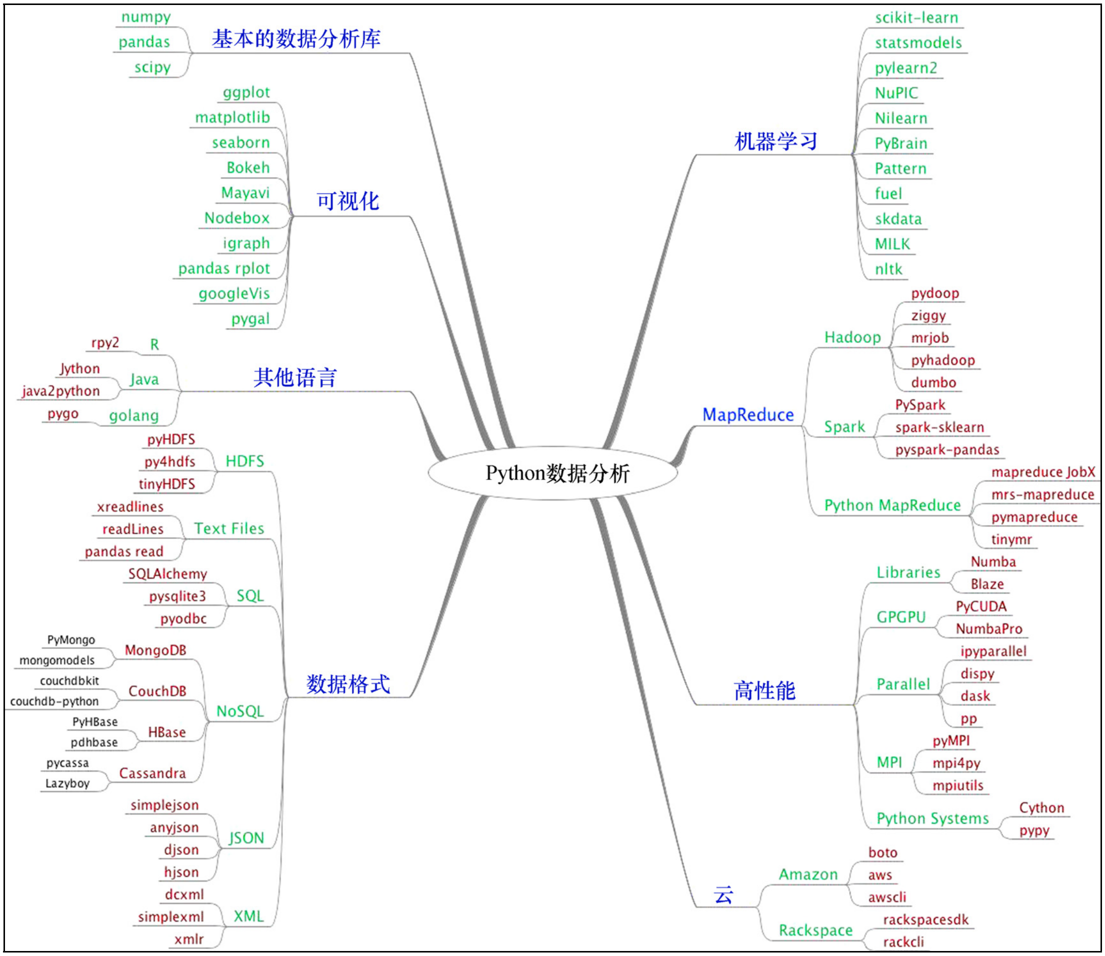


图 1-1

本章将涉及以下主题。

- 安装 Python 3
- 将 IPython 用作 shell
- 阅读手册页
- Jupyter Notebook

- NumPy 数组
- 一个简单的应用
- 何处寻找帮助和参考资料
- 列出 Python 库中的模块
- 利用 matplotlib 可视化数据

1.1 安装 Python 3

本书所用软件都是基于 Python 3 的，所以必须首先安装 Python 3。不过，对于某些操作系统而言，Python 3 是默认安装的。Python 具有多种实现，其中包括具有商业版权的实现和发行版。在本书中，我们只关注标准 Python 实现，因为它与 NumPy 完全兼容。

> **提示**
> 读者可以从 https://www.python.org/download/页面下载 Python 3.5.x。在这个网站上，我们可以找到为 Windows 和 Mac OS X 系统开发的安装程序，以及为 Linux、UNIX 和 Mac OS X 系统提供的源码包。我们可以从 https://docs.python.org/3/using/index.html 上找到在各种操作系统上安装和使用 Python 的相关说明。

本章需要安装的软件，在 Windows、各种 Linux 发行版本和 Mac OS X 系统上都有相应的二进制安装程序。当然，如果读者愿意，也可以使用相应的源代码发行包。对于 Python，要求其版本为 3.5.x 或更高。Python 2.7 版本的支持与维护工作已经从 2015 年延续至 2020 年，之后，我们不得不迁移到 Python 3。

1.1.1 安装数据分析程序库

下面开始介绍如何在 Windows、Linux 和 Mac OS X 上安装和设置 NumPy、SciPy、Pandas、Matplotlib、IPython 和 Jupyter Notebook。下面来详细了解一下这个过程。在本书中，我们将使用 pip3 来安装这些库。因为从 3.4 版本起，pip3 已经默认包含在 Python 的安装程序中了。

1.1.2 Linux 平台或 Mac OS X 平台

为了安装这些基础的程序库，可以运行以下命令。

```
$ pip3 install numpy scipy pandas matplotlib jupyter notebook
```

如果当前登录的账户没有足够权限，则需要在上面的命令行前面添加 sudo。

1.1.3 Windows 平台

在撰写本书时，我们在 Windows 10 虚拟机上安装了以下软件，作为安装这些程序库的先决条件。

- Python 3.6。
- Microsoft Visual C++ Build Tools 2015。

下载并安装适用于 Windows 平台的、预编译好的 NumPy 和 SciPy 二进制文件。

- 我们下载了 numpy-1.12.0 + mkl-cp36-cp36m-win_amd64.whl 和 scipy-0.18.1-cp36-cp36m-win_amd64.whl。
- 下载完成后，执行命令 pip3 install Downloads\numpy-1.12.0 + mkl-cp36-cp36m-win_amd64.whl 和 pip3 install Downloads\scipy-0.18.1-cp36-cp36m-win_amd64.whl。

安装上述软件后，为了安装其余的基础程序库，可以运行以下命令。

```
$ pip3 install pandas matplotlib jupyter
```

> **小技巧**
> 使用这些命令安装 Jupyter 时，要先安装所有必需的软件包，如 Notebook 和 IPython。

1.2 将 IPython 用作 shell

我们知道，科学家、数据分析师和工程师经常需要进行实验，而 IPython 正是为实验而生的。对于 IPython 提供的交互式环境，明眼人一看就知道它与 MATLAB、Mathematica 和 Maple 非常接近。

下面是 IPython shell 的一些特性。

- Tab 补全功能（Tab completion），可以帮助查找命令
- 历史记录机制

- 行内编辑
- 利用%run 调用外部 Python 脚本
- 访问系统命令
- 访问 Python 的调试工具和分析工具

下面给出 IPython shell 的使用方法。

- 启动会话：要想使用 IPython 启动会话，需要在命令行中输入以下指令。

```
$ ipython3
Python 3.5.2 (default, Sep 28 2016, 18:08:09)
Type "copyright", "credits" or "license" for more information.

IPython 5.1.0 -- An enhanced Interactive Python.
?         -> Introduction and overview of IPython's features.
%quickref -> Quick reference.
Help      -> Python's own help system.
object?   -> Details about 'object', use 'object??' for extra
             details.

In [1]: quit()
```

> **小技巧**
> 退出 IPython shell 时，可以使用 quit()函数或者 Ctrl+D 组合键。

- 保存会话：有时我们可能想要恢复之前做过的实验。对于 IPython 来说，这很容易，只要保存了会话，就可以供将来继续使用，具体命令如下。

```
In [1]: %logstart
Activating auto-logging. Current session state plus future
input saved:
Filename : ipython_log.py
Mode : rotate
Output logging : False
Raw input log : False
Timestamping : False State : active
```

使用下列命令可以关闭记录功能。

```
In [9]: %logoff
Switching logging OFF
```

- 执行系统的 shell 命令：在默认情况下，IPython 允许通过在命令前面追加“!”号来执行系统的 shell 命令。举例来说，我们输入下面的命令将会得到当前日期。

```
In [1]: !date
```

事实上，任何前置了“!”号的命令行都将发送给系统的 shell 来处理。此外，可以通过如下方法来存储命令的输出结果。

```
In [2]: thedate = !date
In [3]: thedate
```

- 显示历史上用过的命令：可以利用%hist 命令来显示之前用过的命令，例子如下。

```
In [1]: a = 2 + 2
In [2]: a
Out[2]: 4
In [3]: %hist
a = 2 + 2
a
%hist
```

这在命令行接口（Command Line Interface，CLI）环境中是一种非常普遍的功能。此外，我们还可以用-g 开关在历史命令中进行搜索，例子如下。

```
In [5]: %hist -g a = 2
1: a = 2 + 2
```

在上面的过程中，我们使用了一些所谓的魔力函数（magic functions），这些函数均以“%”开头。当魔力函数单独用于一行时，就可以省略前缀“%”。

1.3 学习手册页

当使用 IPython 导入相应的程序库后，可以通过 help 命令打开 NumPy 函数的手册页——即使不知道该函数的确切名称。我们可以先输入几个字符，然后利用 Tab 键就可以自动补全剩下的字符。下面以 arange()函数为例，说明如何查阅与其有关的资料。

这里给出两种翻阅相关信息的方法。

- 调用 help 函数：输入 help 命令（并输入函数名中的前几个字符，再按 Tab 键。这时将出现一个函数列表（见图 1-2），我们可以通过方向键从函数名列表中进行选

择，然后按 Enter 键进行确认），最后按 Enter 键盘，结束 help 函数的调用。

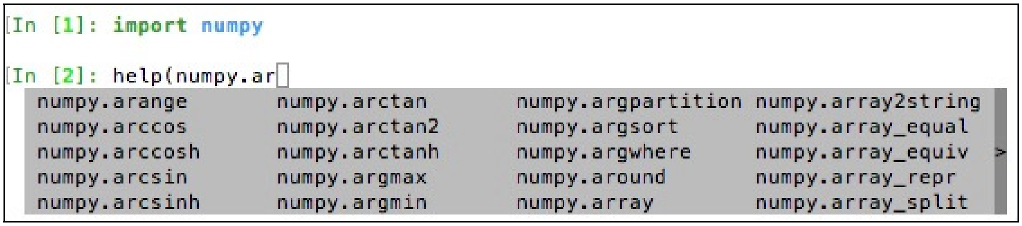


图 1-2

- 通过问号进行查询：另一种方法是在函数名后面加上问号，当然，前提条件是我们已经知道函数名，好处是不必输入 help 命令，例子如下。

```
In [3]: numpy.arange?
```

Tab 补全功能依赖于 readline，所以务必确保先前已经安装了该软件。如果没有安装，可以使用 pip 完成安装，具体命令如下。

```
$ pip3 install readline
```

利用问号，我们可以从文档字符串（docstrings）中获取所需信息。

1.4 Jupyter Notebook

Jupyter Notebook 以前被称为 IPython Notebooks，它提供了一种以特殊格式创建和共享具有文本、图表和 Python 代码的网页的工具。

很多时候，notebook 都是用于演示 Python 软件，或者用作一款教学工具。我们可以单纯使用 Python 代码或者通过特殊的 notebook 格式来导入和导出 notebook。另外，notebook 既可以在本机上跑，也可以放到专用的 notebook 服务器上在线使用。某些云计算解决方案（如 Wakari 和 PiCloud）还支持在云中运行 notebook。云计算的主题将在第 11 章中加以介绍。

为了使用 Jupyter Notebook 启动一个会话，读者可以使用如下命令。

```
$ jupyter-notebook
```

这时将启动 notebook 服务器并打开一个网页，显示该命令所在文件夹的内容。然后，

你可以在 Python 3 中选择 New | Python 3 菜单项来启动一个新的 notebook。

你也可以打开本书的代码包中提供的 ch-01.ipynb。ch-01 是一个 notebook 文件，其中存放了本章中简单应用程序的代码。

1.5 NumPy 数组

安装好 NumPy 后，就可以来看 NumPy 数组了。与 Python 中的列表相比，进行数值运算时 NumPy 数组的效率要高得多。事实上，NumPy 数组是针对某些对象进行了大量的优化工作。

完成相同的运算时，与 Python 代码相比，NumPy 代码用到的显式循环语句明显要少，因为 NumPy 是基于向量化的运算。还记得高等数学中标量和向量的概念吗？例如，数字 2 是一个标量，计算 2+2 时，进行的是标量加法运算。通过一组标量，我们可以构建出一个向量。用 Python 编程的术语来说，我们得到了一个一维数组。当然，这个概念可以扩展至更高的维度。实际上，针对两个数组的诸如加法之类的运算，可以将其转化为一组标量运算。使用纯 Python 时，为了完成该操作，可以使用循环语句遍历第一个数组中的每个元素，并与第二个数组中对应的元素相加。然而，在数学家眼里，这种方法过于繁琐。数学上，可以将这两个向量的加法视为单一操作。实际上，NumPy 数组也可以这么做，而且它用低级 C 例程针对某些操作进行了优化处理，使得这些基本运算效率大为提高。关于 NumPy 数组将在第 2 章中详细介绍。

1.6 一个简单的应用

假设要对向量 a 和 b 进行求和。注意，这里“向量”的含义是数学意义上的，即一个一维数组。在第 4 章中，将遇到一种表示矩阵的特殊 NumPy 数组。向量 a 存放的是整数 0～n−1 的 2 次幂。如果 n 等于 3，那么 a 保存的是 0、1 或 4。向量 b 存放的是整数 0~n 的 3 次幂，所以如果 n 等于 3，那么向量 b 等于 0、1 或者 8。如果使用普通的 Python 代码，该怎么做呢？

在我们想出了一个解决方案后，可以拿来与等价的 NumPy 方案进行比较。

下面的函数没有借助 NumPy，而是使用纯 Python 来解决向量加法问题。

```
def pythonsum(n):
   a = list(range(n))
```

```
    b = list(range(n))
    c = []

    for i in range(len(a)):
        a[i] = i ** 2
        b[i] = i ** 3
        c.append(a[i] + b[i])

    return c
```

下面是利用 NumPy 解决向量加法问题的函数。

```
def numpysum(n):
  a = numpy.arange(n) ** 2
  b = numpy.arange(n) ** 3
  c = a + b
  return c
```

注意，numpysum()无需使用 for 语句。此外，我们使用了来自 NumPy 的 arange()函数，它替我们创建了一个含有整数 0～n 的 NumPy 数组。这里的 arange()函数也是从 NumPy 导入的，所以它加上了前缀 numpy。

现在到了真正有趣的地方。我们在前面讲过，NumPy 在进行数组运算时，速度是相当快的。可是到底有多快呢？下面的程序代码将为我们展示 numpysum()和 pythonsum()这两个函数的实耗时间，这里以μs（微秒）为单位。同时，它还会显示向量 sum 最后面的两个元素值。下面来看使用 Python 和 NumPy 能否得到相同的答案。

```
#!/usr/bin/env/python

import sys
from datetime import datetime
import numpy as np

"""
This program demonstrates vector addition the Python way.
Run the following from the command line:

  python vectorsum.py n

Here, n is an integer that specifies the size of the vectors.

The first vector to be added contains the squares of 0 up to n.
The second vector contains the cubes of 0 up to n.
```

```
The program prints the last 2 elements of the sum and the elapsed time:
"""

def numpysum(n):
   a = np.arange(n) ** 2
   b = np.arange(n) ** 3
   c = a + b

   return c

def pythonsum(n):
   a = list(range(n))
   b = list(range(n))
   c = []

   for i in range(len(a)):
       a[i] = i ** 2
       b[i] = i ** 3
       c.append(a[i] + b[i])

   return c

size = int(sys.argv[1])

start = datetime.now()
c = pythonsum(size)
delta = datetime.now() - start
print("The last 2 elements of the sum", c[-2:])
print("PythonSum elapsed time in microseconds", delta.microseconds)

start = datetime.now()
c = numpysum(size)
delta = datetime.now() - start
print("The last 2 elements of the sum", c[-2:])
print("NumPySum elapsed time in microseconds", delta.microseconds)
```

对于 1000 个、2000 个和 4000 个向量元素，程序的结果如下。

```
$ python3 vectorsum.py 1000
The last 2 elements of the sum [995007996, 998001000]
PythonSum elapsed time in microseconds 976
The last 2 elements of the sum [995007996 998001000]
NumPySum elapsed time in microseconds 87
$ python3 vectorsum.py 2000
The last 2 elements of the sum [7980015996, 7992002000]
```

```
PythonSum elapsed time in microseconds 1623
The last 2 elements of the sum [7980015996 7992002000]
NumPySum elapsed time in microseconds 143
$ python3 vectorsum.py 4000
The last 2 elements of the sum [63920031996, 63968004000]
PythonSum elapsed time in microseconds 3417
The last 2 elements of the sum [63920031996 63968004000]
NumPySum elapsed time in microseconds 237
```

显而易见，NumPy 的运行速度比等价的常规 Python 代码要快很多。有一件事情是肯定的：无论是否使用 NumPy，计算结果都是相同的。但是结果的显示形式还是有所差别的，numpysum()函数给出的结果不包含逗号。为什么会这样？别忘了，我们处理的不是 Python 的列表，而是一个 NumPy 数组。有关 NumPy 数组的更多内容，将在第 2 章中详细介绍。

1.7 从何处寻求帮助和参考资料

表 1-1 列出了在本章中讨论过的 Python 数据分析库的文档网站。

表 1-1

软件包	描述
NumPy 和 SciPy	NumPy 和 SciPy 的主要文档网站是 http://docs.scipy.org/doc/。通过该网站，您可以浏览 NumPy 和 SciPy 程序库的用户指南和参考指南，以及一些相关教程
Pandas	http://pandas.pydata.org/pandas-docs/stable/
Matplotlib	http://matplotlib.org/contents.html
IPython	http://ipython.readthedocs.io/en/stable/
Jupyter Notebook	http://jupyter-notebook.readthedocs.io/en/latest/

流行的软件开发论坛 Stack Overflow 上也有数以百计的 NumPy、SciPy、Pandas、Matplotlib、IPython 和 Jupyter Notebook 方面的讨论。如果读者对这些内容感兴趣，建议进一步学习。

如果你遇到了比较棘手的问题，或者想要持续关注这些程序库的开发进展，可以订阅相应的讨论邮寄列表。订阅后，每天收到的数量不一的邮件，开发者会积极报告这些库的开发进展并热心回答其中的问题。

对于 IRC 用户，可以在 irc://irc.freenode.net 找到一个相关的频道，虽然该频道的名字

是#scipy，但是这并不妨碍我们提问 NumPy 方面的问题，因为 SciPy 用户一般比较熟悉 NumPy，毕竟 SciPy 是以 NumPy 为基础的。在这个 SciPy 频道中，通常有 50 多位成员保持在线。

1.8 查看 Python 库中包含的模块

ch-01.ipynb 文件包含用于查看 NumPy、SciPy、Pandas 和 Matplotlib 库中的模块的代码。现在，读者不用担心这些代码的含义，只要尝试运行一下它们就行了。您可以修改其中的代码以查看其他库中的模块。

1.9 通过 Matplotlib 实现数据的可视化

我们将在后面的章节中详细介绍数据的可视化问题。现在，我们先来加载两个样本数据集并生成一些简单的图形。首先，我们使用以下命令将 sklearn 库安装到要提供加载数据的地方。

```
$ pip3 install scikit-learn
```

接下来，使用以下命令导入数据集。

```
from sklearn.datasets import load_iris
from sklearn.datasets import load_boston
```

然后，导入 Matplotlib 绘图模块。

```
from matplotlib import pyplot as plt
%matplotlib inline
```

加载 iris 数据集，显示数据集的相关描述，同时将第 1 列（萼片长度）作为 x 坐标值，将第 2 列（萼片宽度）作为 y 坐标值。

```
iris = load_iris()
print(iris.DESCR)
data=iris.data
plt.plot(data[:,0],data[:,1],".")
```

这时，将生成如图 1-3 所示的图形。

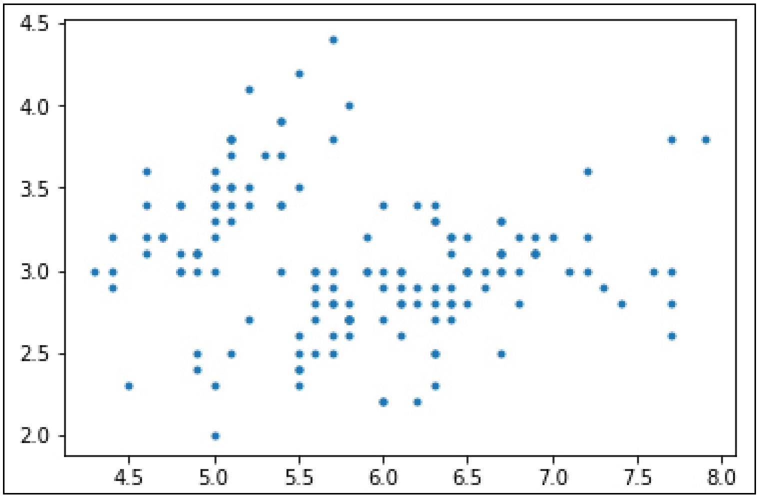


图 1-3

加载波士顿数据集，显示数据集的相关描述，同时将第 3 列（非零售业务的比例）作为 x 坐标值，将第 5 列（一氧化氮浓度）做为 y 坐标值，图上的每个点用“+”号表示。

```
boston = load_boston()
print(boston.DESCR)
data=boston.data
plt.plot(data[:,2],data[:,4],"+")
```

最终得到的图形如图 1-4 所示。

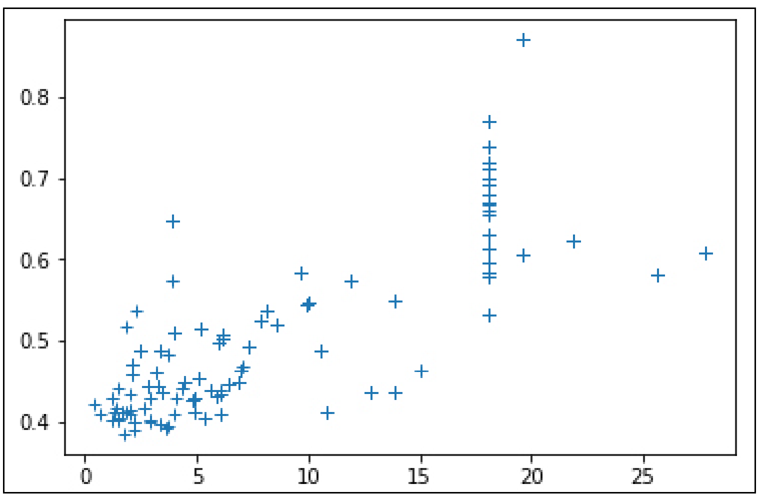


图 1-4

1.10 小结

本章安装了以后要用到的 NumPy、SciPy、Pandas、Matplotlib、IPython 和 Jupyter Notebook 等程序库，并通过一个向量加法程序，体验了 NumPy 带来的卓越性能。此外，我们还探讨了有关的文档和在线资源。同时，我们还尝试通过运行代码来查找库中的模块，并加载了一些样本数据集，还使用 Matplotlib 绘制一些简单的图形。

第 2 章将揭晓与 NumPy 有关的内容，以探索数组和数据类型等基本概念。

第 2 章
NumPy 数组

在前面我们已经尝试了一个基于 SciPy 栈中的基础数据分析程序库的实际代码。在本章中，我们将带领大家一起学习 NumPy 数组，熟悉 NumPy 数组的基础知识。阅读本章后，你会对 NumPy 数组及其相关函数有基本了解。

本章涉及的主题如下。

- NumPy 数组对象
- 创建多维数组
- 选择 NumPy 数组元素
- NumPy 数值类型
- 一维切片和索引
- 改变阵列形状
- 创建阵列视图和副本
- 花式索引
- 使用位置列表进行索引
- 用布尔值索引 NumPy 数组
- NumPy 数组的广播技术

你可以在 Jupyter Notebook 中打开 ch-02.ipynb 文件，按照本章中的示例进行操作，此外也可以新建一个 notebook，手动输入相应的代码。

2.1 NumPy 数组对象

NumPy 提供了一个名为 ndarray 的多维数组对象。NumPy 数组是具有固定大小的类型化数组。Python 的列表是异构的，因此列表的元素可以包含任何对象类型，而 NumPy 数组是同质的，只能存放同一种类型的对象。数组由两部分组成，分别如下。

- 存储在连续的内存块中的实际数据
- 描述实际数据的元数据

实际数据存储在连续的内存块中，因此当大型数据集作为 ndarray 进行加载时，我们就要看有没有足够多的连续内存块了。NumPy 中的大部分数组方法和函数都不会直接修改实际数据，而只能修改元数据。

在之前的章节中，我们曾经用 arange()函数来生成数组。实际上，那是用来存放一组数值的一维数组，这里的 ndarray 则可以具有一个以上的维度。

NumPy 数组的优势

NumPy 数组通常是同质的（有一种特殊的记录数组类型，它可以是异质的），即数组中的数据项的类型必须一致。NumPy 数组元素类型一致的好处是：因为知道数组元素的类型相同，所以能轻松确定存储数组所需空间的大小。同时，NumPy 数组还能够运用向量化运算来处理整个数组；而完成同样的任务，Python 的列表则通常需要借助循环语句遍历列表并对逐个元素进行相应的处理。NumPy 数组的索引方法与 Python 类似，下标从 0 开始。此外，NumPy 使用了优化过的 C API，所以运算速度格外快。

今后，我们会经常利用 arange()子例程来建立数组。本章中的代码片断大部分取自 Jupyter Notebook 会话，而且在这些对话中，NumPy 已经利用 import numpy as np 命令导入进来了。以下代码展示了如何获得数组的数据类型。

```
In: a = np.arange(5)
In: a.dtype
Out: dtype('int64')
```

以上数组的数据类型为 int64（至少在作者的电脑上是这样的），不过如果你的 Python 为 32 位版本的话，得到的结果将是 int32。无论上面哪一种情况，都是在处理整型变量（64 位或者 32 位）。对于数组，除了要知道数据类型外，我们还要注意其形状，这一点非常重要。

在第 1 章中，我们曾经举例说明向量（一维 NumPy 数组）的创建方法。数学家会经常用到向量，对我们来说，最常用的却是更高维度的对象。下面来看刚刚生成的那个向量的形状。

```
In: a
Out: array([0, 1, 2, 3, 4])
In: a.shape
Out: (5,)
```

如你所见，该向量有 5 个元素，它们的值分别是从 0~4。该数组的 shape 属性是一个元组，就本例而言，这是一个单元素元组，存放的是数组在每一个维度的长度。

2.2 创建多维数组

既然我们已经知道创建向量的方法，下面开始学习如何建立多维 NumPy 数组。生成矩阵后，我们再来看它的形状，代码如下。

（1）我们创建多维数组，代码如下。

```
In: m = np.array([np.arange(2), np.arange(2)])
In: m
Out:
array([[0, 1],
       [0, 1]])
```

（2）显示该数组的形状的代码如下。

```
In: m.shape
Out: (2, 2)
```

上面我们用 arange()子例程直接建立了一个 2×2 的数组，而利用 array()函数创建数组时，则需要传递给它一个对象，同时这个对象还必须是数组类型的，如 Python 的列表。在上面的例子中，我们传给它的是由两个数组组成的一个列表。该对象是 array()函数唯一所需的参数，而 NumPy 的函数往往有多个可选参数，而且这些参数都带有预定义的缺省选项。

2.3 选择 NumPy 数组元素

有时我们可能想从数组中选择指定的元素。如何做到这一点呢？我们不妨从创建一个 2×2 矩阵着手。

```
In: a = np.array([[1,2],[3,4]])
```

```
In: a
Out:
array([[1, 2],
       [3, 4]])
```

上面的矩阵是通过向 array()函数传递一个由列表组成的列表得到的。接下来我们要逐个选择矩阵的各个元素，代码如下。别忘了下标是从 0 开始的。

```
In: a[0,0]
Out: 1
In: a[0,1]
Out: 2
In: a[1,0]
Out: 3
In: a[1,1]
Out: 4
```

可见，选择数组元素是一件非常简单的事情。对于数组 a，我们只要通过 a[m,n]的形式，就能访问数组内的元素，其中 m 和 n 为数组元素的下标。数组元素的下标如图 2-1 所示。

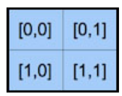


图 2-1

2.4 NumPy 的数值类型

Python 自身虽然支持整型、浮点型和复数型，但对于科学计算来说，还远远不够。现实中，我们仍然需要更多的数据类型来满足在精度和存储大小方面的各种不同要求。为此，NumPy 提供了更加丰富的数据类型。注意，NumPy 中跟数学运算有关的数据类型的名称都以数字结尾。而这个数字指示了该类型的变量所占用的二进制位数。表 2-1（改编自《NumPy 用户指南》）概述了 NumPy 的各种数值类型。

表 2-1

类型	说明
bool	布尔型（值为 True 或 False），占用 1bit
inti	其长度取决于平台的整数（通常为 int32 或者 int64）
int8	字节类型（取值范围从−128～127）

续表

类型	说明
int16	整型（取值范围−32768～32767）
int32	整型（取值范围为-2^{31}～$2^{31}-1$）
int64	整型（取值范围为-2^{63}～$2^{63}-1$）
uint8	无符号整型（取值范围为 0～255）
uint16	无符号整型（取值范围为 0～65535）
uint32	无符号整型（取值范围为 0～$2^{32}-1$）
uint64	无符号整型（取值范围为 0～$2^{64}-1$）
float16	半精度浮点型：符号占用 1bit，指数占用 5 bit，尾数占用 10 bit
float32	单精度浮点型：符号占用 1 bit，指数占用 8 bit，尾数占用 23 bit
float64 或者 float	双精度浮点型：符号占用 1 bit，指数占用 11 bit，尾数占用 52 bit
complex64	复数类型，由两个 32 位浮点数（实部和虚部）表示
complex128 或者 complex	复数类型，由两个 64 位浮点数（实部和虚部）表示

每一种数据类型都有相应的转换函数，写法如下。

```
In: np.float64(42)
Out: 42.0
In: np.int8(42.0)
Out: 42
In: np.bool(42)
Out: True
In: np.bool(0)
Out: False
In: np.bool(42.0)
Out: True
In: np.float(True)
Out: 1.0
In: np.float(False)
Out: 0.0
```

许多函数都带有一个指定数据类型的参数，该参数通常是可选的。

```
In: np.arange(7, dtype= np.uint16)
Out: array([0, 1, 2, 3, 4, 5, 6], dtype=uint16)
```

切记：不能把复数类型转化成整型。当我们试图进行这种转换时，将会触发 TypeError

错误，就像下面这样。

```
In: np.int(42.0 + 1.j)Traceback (most recent call last):
<ipython-input-24-5c1cd108488d> in <module>()
----> 1 np.int(42.0 + 1.j)
TypeError: can't convert complex to int
```

同样，也不允许把复数转化成浮点数。另外，复数的分量 j 是其虚部的系数。不过，允许把浮点数转换成复数，如 complex(1.0)是合法的。复数的实部和虚部分别使用 real()函数和 imag()函数提取。

2.4.1 数据类型对象

数据类型对象是 numpy.dtype 类的实例。再强调一次，数组也有数据类型。严格地讲，NumPy 数组中的每个元素都要具有相同的数据类型。数据类型对象表明了数据占用的字节数。所占用字节的具体数目一般存放在类 dtype 的 itemsize 属性中。

```
In: a.dtype.itemsize
Out: 8
```

2.4.2 字符码

NumPy 之所以提供字符码，是为了与其前身 Numeric 向后兼容。一般我们不建议使用字符码，这里为什么又提供这些代码呢？因为我们会在许多地方碰到它们，但是，编写代码时我们应当使用 dtype 对象。表 2-2 展示了常见的一些数据类型及其相应的字符码。

表 2-2

类型	字符码
整型	i
无符号整型	u
单精度浮点型	f
双精度浮点型	d
布尔型	b
复数型	D
字符串	S
万国码（unicode）	U
空类型（Void）	V

下面的代码将生成一个单精度浮点型的数组。

```
In: arange(7, dtype='f')
Out: array([ 0., 1., 2., 3., 4., 5., 6.], dtype=float32)
```

类似地，下列代码将创建一个负数类型的数组。

```
In: arange(7, dtype='D')
Out: array([ 0.+0.j, 1.+0.j, 2.+0.j, 3.+0.j, 4.+0.j, 5.+0.j,
6.+0.j])
```

2.4.3 dtype 构造函数

创建数据类型时手段有很多，下面我们以浮点型数据为例进行说明（以下代码取自本书代码包中的 dtypeconstructors.py 文件）。

- 我们可以用 Python 自带的常规浮点型，代码如下。

```
In: np.dtype(float)
Out: dtype('float64')
```

- 我们可以用字符码规定单精度浮点数，代码如下。

```
In: np.dtype('f')
Out: dtype('float32')
```

- 我们可以用字符码定义双精度浮点数，代码如下。

```
In: np.dtype('d')
Out: dtype('float64')
```

- 我们可以向 dtype 构造函数传递一个双字符码。其中，第一个字符表示数据类型，第二个字符是一个数字，表示该类型占用的字节数（数字 2、4 和 8 分别对应 16 位、32 位和 64 位浮点数）。

```
In: np.dtype('f8')
Out: dtype('float64')
```

我们可以通过 sctypeDict.keys()函数列出所有数据类型的字符码，代码如下。

```
In: np.sctypeDict.keys()
Out: dict_keys(['?', 0, 'byte', 'b', 1, 'ubyte', 'B', 2, 'short', 'h', 3,
'ushort', 'H', 4, 'i', 5, 'uint', 'I', 6, 'intp', 'p', 7, 'uintp', 'P', 8,
```

```
'long', 'l', 'L', 'longlong', 'q', 9, 'ulonglong', 'Q', 10, 'half', 'e',
23, 'f', 11, 'double', 'd', 12, 'longdouble', 'g', 13, 'cfloat', 'F', 14,
'cdouble', 'D', 15, 'clongdouble', 'G', 16, 'O', 17, 'S', 18, 'unicode',
'U', 19, 'void', 'V', 20, 'M', 21, 'm', 22, 'bool8', 'Bool', 'b1',
'float16', 'Float16', 'f2', 'float32', 'Float32', 'f4', 'float64',
'Float64', 'f8', 'float128', 'Float128', 'f16', 'complex64', 'Complex32',
'c8', 'complex128', 'Complex64', 'c16', 'complex256', 'Complex128', 'c32',
'object0', 'Object0', 'bytes0', 'Bytes0', 'str0', 'Str0', 'void0', 'Void0',
'datetime64', 'Datetime64', 'M8', 'timedelta64', 'Timedelta64', 'm8',
'int64', 'uint64', 'Int64', 'UInt64', 'i8', 'u8', 'int32', 'uint32',
'Int32', 'UInt32', 'i4', 'u4', 'int16', 'uint16', 'Int16', 'UInt16', 'i2',
'u2', 'int8', 'uint8', 'Int8', 'UInt8', 'i1', 'u1', 'complex_', 'int0',
'uint0', 'single', 'csingle', 'singlecomplex', 'float_', 'intc', 'uintc',
'int_', 'longfloat', 'clongfloat', 'longcomplex', 'bool_', 'unicode_',
'object_', 'bytes_', 'str_', 'string_', 'int', 'float', 'complex', 'bool',
'object', 'str', 'bytes', 'a'])
```

2.4.4 dtype 属性

类 dtype 提供了许多有用的属性，如可以通过 dtype 的属性获取某种数据类型对应的字符码。

```
In: t = np.dtype('Float64')
In: t.char
Out: 'd'
```

类型属性相当于数组元素对象的类型，如下所示。

```
In: t.type
Out: numpy.float64
```

dtype 的属性 str 中保存的是一个表示数据类型的字符串，其中第一个字符描述字节顺序，如果需要，后面会跟着字符码和数字，用来表示存储每个数组元素所需的字节数。这里，字节顺序（endianness）规定了 32 位或 64 位字内部各个字节的存储顺序。对于大端（big-endian）顺序，先存放权重最高的字节，用符号“>”指出；当使用小端（little-endian）顺序时，则存放权重最低的字节，用符号“<”指出。下面以代码为例进行说明。

```
In: t.str
Out: '<f8'
```

2.5 一维数组的切片与索引

一维 NumPy 数组的切片操作与 Python 列表的切片一样。下面先来定义包含数字 0、1、2, …, 8 的一个数组，然后通过指定下标 3～7 来选择该数组的部分元素，这实际上就是提取数组中值为 3～6 的那些元素。

```
In: a = np.arange(9)
In: a[3:7]
Out: array([3, 4, 5, 6])
```

我们可以用下标选择元素，下标范围从 0～7，而且下标每次递增 2，如下所示。

```
In: a[:7:2]
Out: array([0, 2, 4, 6])
```

正如使用 Python 那样，我们也可用负值下标来反转数组。

```
In: a[::-1]
Out: array([8, 7, 6, 5, 4, 3, 2, 1, 0])
```

2.6 处理数组形状

前面我们学习过 reshape()函数，实际上，除了数组形状的调整外，数组的扩充也是一个经常碰到的乏味工作。比如，我们可以想象一下将多维数组转换成一维数组时的情形。我们创建一个数组 b，下面的多个例子都会用到它。

```
In: b = np.arange(24).reshape(2,3,4)

In: print(b)

Out: [[[ 0, 1, 2, 3],
        [ 4, 5, 6, 7],
        [ 8, 9, 10, 11]],

       [[12, 13, 14, 15],
        [16, 17, 18, 19],
        [20, 21, 22, 23]]]
```

我们可以利用以下函数处理数组的形状。

- 拆解：可以用 ravel()函数将多维数组变成一维数组，代码如下。

```
In: b
Out:
array([[[ 0,  1,  2,  3],
        [ 4,  5,  6,  7],
        [ 8,  9, 10, 11]],
       [[12, 13, 14, 15],
        [16, 17, 18, 19],
        [20, 21, 22, 23]]])
In: b.ravel()
Out:
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9,  10,  11,  12,  13,
14, 15, 16, 17, 18, 19, 20, 21, 22, 23])
```

- 拉直（Flatten）：flatten()函数的名字取得非常贴切，其功能与 ravel()相同。可是，flatten()返回的是真实的数组，需要分配新的内存空间；而 ravel()函数返回的只是数组的视图。这意味着，我们可以像下面这样直接操作数组。

```
In: b.flatten()
Out:
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9,  10,  11,  12,  13,
14, 15, 16, 17, 18, 19, 20, 21, 22, 23])
```

- 用元组指定数组形状：除 reshape()函数外，还可以用元组来轻松定义数组的形状，代码如下。

```
In: b.shape = (6,4)
In: b
Out:
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11],
       [12, 13, 14, 15],
       [16, 17, 18, 19],
       [20, 21, 22, 23]])
```

- 可见，上述代码直接改变了数组的形状。这样，我们就得到了一个 6×4 的数组。
- 转置：在线性代数中，矩阵的转置操作非常常见。转置是一种数据变换方法，对于二维表而言，转置就意味着行变成列，同时列变成行。转置也可以通过下列代码完成。

```
In: b.transpose()
Out:
```

```
array([[ 0, 4,  8, 12, 16, 20],
       [ 1, 5,  9, 13, 17, 21],
       [ 2, 6, 10, 14, 18, 22],
       [ 3, 7, 11, 15, 19, 23]])
```

- 调整大小：函数 resize()的作用类似于 reshape()，但是会改变所作用的数组。

```
In: b.resize((2,12))
In: b
Out:
array([[ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11],
       [12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23]])
```

2.6.1 堆叠数组

从深度看，数组既可以横向叠放，也可以竖向叠放。为此，可以使用 vstack()、dstack()、hstack()、column_stack()、row_stack()和 concatenate()等函数。在此之前，我们先要建立某些数组。

```
In: a = np.arange(9).reshape(3,3)
In: a
Out:
array([[0, 1, 2],
       [3, 4, 5],
       [6, 7, 8]])
In: b = 2 * a
In: b
Out:
array([[ 0,  2,  4],
       [ 6,  8, 10],
       [12, 14, 16]])
```

就像前面所说的，可以用下列技术来堆放数组。

- 水平叠加：先介绍水平叠加方式，即用元组确定 ndarrays 数组的形状，然后交给 hstack()函数来码放这些数组，具体如下。

```
In: np.hstack((a, b))
Out:
array([[ 0, 1, 2,  0,  2,  4],
       [ 3, 4, 5,  6,  8, 10],
       [ 6, 7, 8, 12, 14, 16]])
```

- 用 concatenate()函数也能达到同样的效果，代码如下。

```
In: np.concatenate((a, b), axis=1)
Out:
array([[ 0, 1, 2,  0,  2,  4],
       [ 3, 4, 5,  6,  8, 10],
       [ 6, 7, 8, 12, 14, 16]])
```

水平叠加过程的示意图如图 2-2 所示。

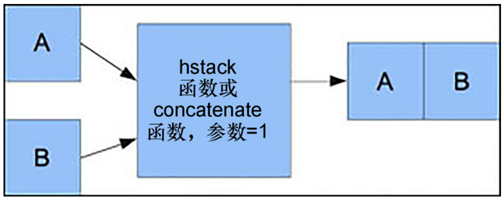


图 2-2

- 垂直叠加：使用垂直叠加方法时，先要构建一个元组，然后将元组交给 vstack()函数来码放数组，代码如下。

```
In: np.vstack((a, b))
Out:
array([[ 0,  1,  2],
       [ 3,  4,  5],
       [ 6,  7,  8],
       [ 0,  2,  4],
       [ 6,  8, 10],
       [12, 14, 16]])
```

当参数 axis 设置为 0 时，concatenate()函数也会得到同样的效果。实际上，这是该参数的缺省值，代码如下。

```
In: np.concatenate((a, b), axis=0)
Out:
array([[ 0,  1,  2],
     [ 3,  4,  5],
     [ 6,  7,  8],
     [ 0,  2,  4],
     [ 6,  8, 10],
     [12, 14, 16]])
```

垂直叠加过程的示意图如图 2-3 所示。

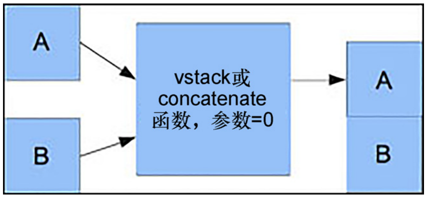


图 2-3

- 深度叠加：除此之外，还有一种深度叠加方法，这要用到 dstack()函数和一个元组。这种方法是沿着第三个坐标轴（纵向）的方向来叠加一摞数组。举例来说，我们可以在一个图像数据的二维数组上叠加另一幅图像的数据，代码如下。

```
In: np.dstack((a, b))
Out:
array([[[ 0,  0],
        [ 1,  2],
        [ 2,  4]],
       [[ 3,  6],
        [ 4,  8],
        [ 5, 10]],
       [[ 6, 12],
        [ 7, 14],
        [ 8, 16]]])
```

- 列式堆叠：column_stack()函数以列方式对一维数组进行堆叠，代码如下。

```
In: oned = np.arange(2)
In: oned
Out: array([0, 1])
In: twice_oned = 2 * oned
In: twice_oned
Out: array([0, 2])
In: np.column_stack((oned, twice_oned))
Out:
array([[0, 0],
       [1, 2]])
```

用这种方法堆叠二维数组时，过程类似于 hstack()函数，代码如下。

```
In: np.column_stack((a, b))
Out:
array([[ 0, 1, 2,  0,  2,  4],
       [ 3, 4, 5,  6,  8, 10],
       [ 6, 7, 8, 12, 14, 16]])
```

```
In: np.column_stack((a, b)) == np.hstack((a, b))
Out:
array([[ True, True, True, True, True, True],
       [ True, True, True, True, True, True],
       [ True, True, True, True, True, True]],
       dtype=bool)
```

是的，你猜得没错！我们用“==”运算符对两个数组进行了比对。

- 行式堆叠：同时，NumPy 自然也有以行方式对数组进行堆叠的函数，这个用于一维数组的函数名为 row_stack()，它将数组作为行码放到二维数组中，代码如下。

```
In: np.row_stack((oned, twice_oned))
Out:
array([[0, 1],
       [0, 2]])
```

对于二维数组，row_stack()函数相当于 vstack()函数，代码如下。

```
In: np.row_stack((a, b))
Out:
array([[ 0,  1,  2],
       [ 3,  4,  5],
       [ 6,  7,  8],
       [ 0,  2,  4],
       [ 6,  8, 10],
       [12, 14, 16]])
In: np.row_stack((a,b)) == np.vstack((a, b))
Out:
array([[ True, True, True],
       [ True, True, True],
       [ True, True, True],
       [ True, True, True],
       [ True, True, True],
       [ True, True, True]], dtype=bool)
```

2.6.2　拆分 NumPy 数组

我可以从纵向、横向和深度方向来拆分数组，相关函数有 hsplit()、vsplit()、dsplit()和 split()。我们既可以把数组分成相同形状的数组，也可以从规定的位置开始切取数组。下面对相关函数逐个详解。

- 横向拆分：对于一个 3×3 数组，可以沿着横轴方向将其分解为 3 部分，并且各部

分的大小和形状完全一致，代码（它取自本书代码包中的 splitting.py 文件）如下。

```
In: a
Out:
array([[0, 1, 2],
       [3, 4, 5],
       [6, 7, 8]])
In: np.hsplit(a, 3)
Out:
[array([[0],
       [3],
       [6]]),
 array([[1],
       [4],
       [7]]),
 array([[2],
       [5],
       [8]])]
```

这相当于调用了参数 axis=1 的 split()函数。

```
In: np.split(a, 3, axis=1)
Out:
[array([[0],
       [3],
       [6]]),
 array([[1],
       [4],
       [7]]),
 array([[2],
       [5],
       [8]])]
```

- 纵向拆分：vsplit()函数将沿着纵轴方向分解数组。

```
In: np.vsplit(a, 3)
Out: [array([[0, 1, 2]]), array([[3, 4, 5]]),
      array([[6, 7, 8]])]
```

当参数 axis=0 时，split()函数也会沿着纵轴方向分解数组，代码如下

```
In: np.split(a, 3, axis=0)
Out: [array([[0, 1, 2]]), array([[3, 4, 5]]),
      array([[6, 7, 8]])]
```

- 深向拆分：dsplit()函数会沿着深度方向分解数组。下面以秩为 3 的数组为例进行说明。

```
In: c = np.arange(27).reshape(3, 3, 3)
In: c
Out:
array([[[ 0,  1,  2],
        [ 3,  4,  5],
        [ 6,  7,  8]],
       [[ 9, 10, 11],
        [12, 13, 14],
        [15, 16, 17]],
       [[18, 19, 20],
        [21, 22, 23],
        [24, 25, 26]]])
In: np.dsplit(c, 3)
Out:
[array([[[ 0],
        [ 3],
        [ 6]],
       [[ 9],
        [12],
        [15]],
       [[18],
        [21],
        [24]]]),
array([[[ 1],
        [ 4],
        [ 7]],
       [[10],
        [13],
        [16]],
       [[19],
        [22],
        [25]]]),
array([[[ 2],
        [ 5],
        [ 8]],
       [[11],
        [14],
        [17]],
       [[20],
        [23],
        [26]]])]
```

2.6.3 NumPy 数组的属性

下面举例说明 NumPy 数组各种属性的详细用法。我们创建一个数组 b，以便后面的例

子使用。

```
In: b = np.arange(24).reshape(2, 12)
In: b
Out:
array([[ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11],
       [12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23]])
```

除 shape 和 dtype 属性外，ndarray 类型的属性还很多，下面逐一列出。

- ndim 属性存储的是维度的数量，下面举例说明。

```
In: b.ndim
Out: 2
```

- size 属性用来保存元素的数量，用法如下。

```
In: b.size
Out: 24
```

- itemsize 属性可以返回数组中各个元素所占用的字节数，代码如下。

```
In: b.itemsize
Out: 8
```

- 如果想知道存储整个数组所需的字节数量，可以求助于 nbytes 属性。这个属性的值正好是 itemsize 属性值和 size 属性值之积。

```
In: b.nbytes
Out: 192
In: b.size * b.itemsize
Out: 192
```

- T 属性的作用与 transpose()函数相同，下面举例说明。

```
In: b.resize(6,4)
In: b
Out:
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11],
       [12, 13, 14, 15],
       [16, 17, 18, 19],
       [20, 21, 22, 23]])
```

```
In: b.T
Out:
array([[ 0, 4,  8, 12, 16, 20],
       [ 1, 5,  9, 13, 17, 21],
       [ 2, 6, 10, 14, 18, 22],
       [ 3, 7, 11, 15, 19, 23]])
```

- 如果数组的秩（rank）小于 2，那么所得只是一个数组的视图。

```
In: b.ndim
Out: 1
In: b.T
Out: array([0, 1, 2, 3, 4])
```

- 对于 NumPy 来说，复数用 j 表示，下面举例说明如何用复数生成一个数组。

```
In: b = np.array([1.j + 1, 2.j + 3])
In: b
Out: array([ 1.+1.j, 3.+2.j])
```

- real 属性将返回数组的实部；当数组元素全为实数时，就返回数组本身，代码如下。

```
In: b.real
Out: array([ 1., 3.])
```

- imag 属性存放的是数组的虚部。

```
In: b.imag
Out: array([ 1., 2.])
```

- 如果数组含有复数，那么它的数据类型将自动变为复数类型，代码如下。

```
In: b.dtype
Out: dtype('complex128')
In: b.dtype.str
Out: '<c16'
```

- flat 属性可返回一个 numpy.flatiter 对象，这是获得 flatiter 对象的唯一方法，但我们无法访问 flatiter 的构造函数。我们可以使用 flat 的迭代器来遍历数组，就像遍历“胖”数组那样，代码如下。

```
In: b = np.arange(4).reshape(2,2)
```

```
In: b
Out:
array([[0, 1],
       [2, 3]])
In: f = b.flat
In: f
Out: <numpy.flatiter object at 0x103013e00>
In: for item in f: print(item)
Out:
0
1
2
3
```

当然，取得 flatiter 对象的元素也不难，代码如下。

```
In: b.flat[2]
Out: 2
```

此外，还可以请求多个元素，代码如下。

```
In: b.flat[[1,3]]
Out: array([1, 3])
```

同时，还可以给 flat 属性赋值。不过，需要注意的是，这个值将会覆盖整个数组内所有元素的值，下面举例说明。

```
In: b.flat = 7
In: b
Out:
array([[7, 7],
       [7, 7]])
```

此外，还可以返回指定的元素，代码如下。

```
In: b.flat[[1,3]] = 1
In: b
Out:
array([[7, 1],
       [7, 1]])
```

图 2-4 是对 ndarray 各种属性的一个小结。

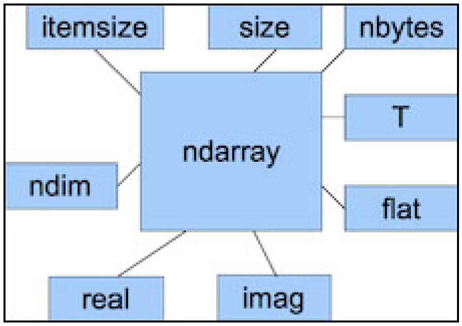


图 2-4

2.6.4 数组的转换

我们可以把 NumPy 数组转换成 Python 列表，使用 tolist()函数即可。下面简单解释一下。

- 转换成列表：

```
In: b
Out: array([ 1.+1.j,  3.+2.j])
In: b.tolist()
Out: [(1+1j), (3+2j)]
```

- astype()函数可以把数组元素转换成指定类型，代码如下。

```
In: b
Out: array([ 1.+1.j,  3.+2.j])
In: b.astype(int)
/usr/local/lib/python3.5/site-packages/ipykernel/__main__.py:1:
ComplexWarning: Casting complex values to real discards the
imaginary part
...
Out: array([1, 3])
In: b.astype('complex')
Out: array([ 1.+1.j,  3.+2.j])
```

提示

当 complex 类型转换成 int 类型时，虚部将被丢弃。另外，我们还需要将数据类型的名称以字符串的形式传递给 astype()函数。

上述代码没有显示警告信息，因为这次使用的是正确的数据类型。

2.7 创建数组的视图和拷贝

在介绍 ravel()函数的示例中，我们提到了视图的概念。不过，请不要与数据库中的视图概念混淆。在 NumPy 的世界里，视图不是只读的，因为你不可能守着基础数据一动不动。关键在于要知道当前处理的是共享的数组视图还是数组数据的副本。举例来说，我们可以取数组的一部分来生成视图。这意味着，如果先将数组的某部分赋给一个变量，然后修改原数组中相应位置的数据，那么这个变量的值也会随之变化。我们可以根据 SciPy 包中的面部照片来创建数组，然后创建视图，随后修改它。这里，莱娜肖像的数组是从 SciPy 函数获得的。

（1）获取面部图片。

```
face = scipy.misc.face()
```

（2）为该面部数组创建副本。

```
acopy = face.copy()
```

（3）为该数组创建一个视图。

```
aview = face.view()
```

（4）通过 flat 迭代器将视图中所有的值全部设为 0。

```
aview.flat = 0
```

最后，只有一幅图片可以看到图像，而其他图片根本看不到什么，如图 2-5 所示。

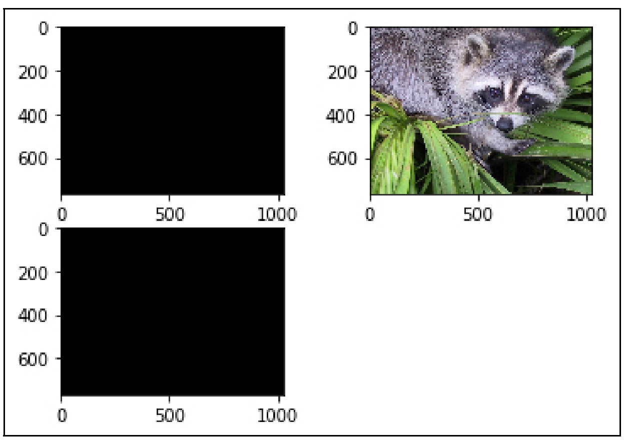


图 2-5

下面的代码很好地展示了数组的视图和副本的特点。

```
import scipy.misc
import matplotlib.pyplot as plt

face = scipy.misc.face()
acopy = face.copy()
aview = face.view()
aview.flat = 0
plt.subplot(221)
plt.imshow(face)
plt.subplot(222)
plt.imshow(acopy)
plt.subplot(223)
plt.imshow(aview)
plt.show()
```

可见，在程序结束部分修改视图，同时改变了原来的莱娜数组。这导致 3 副图片全部变蓝（如果阅读的是本书的印刷版，也可能显示为黑色），复制的数组则没有任何变化。所以一定要记住：视图不是只读的。

2.8 花式索引

花式索引是一种传统的索引方法，它不使用整数或者切片。这里，我们将利用花式索引把莱娜照片对角线上的值全部设置为 0，相当于沿着两条交叉的对角线画两条黑线。

下面我们给出本例中的相应代码。

```
import scipy.misc
import matplotlib.pyplot as plt

face = scipy.misc.face()
xmax = face.shape[0]
ymax = face.shape[1]
face=face[:min(xmax,ymax),:min(xmax,ymax)]
xmax = face.shape[0]
ymax = face.shape[1]
face[range(xmax), range(ymax)] = 0
face[range(xmax-1,-1,-1), range(ymax)] = 0
plt.imshow(face)
plt.show()
```

下面我们对上述代码进行简单说明。

（1）将第一条对角线上的值设为 0。

为了将对角线上的值设置为 0，我们需要给 x 和 y 值（直角坐标系中的坐标）规定两个不同的范围。

```
face[range(xmax), range(ymax)] = 0
```

（2）将另一条对角线上的值设为 0。

要设置另一条对角线上的值，需要规定两个不同的取值范围，但是规则不变。

```
face[range(xmax-1,-1,-1), range(ymax)] = 0
```

划掉相片对角线后，最后得到图 2-6 所示的效果。

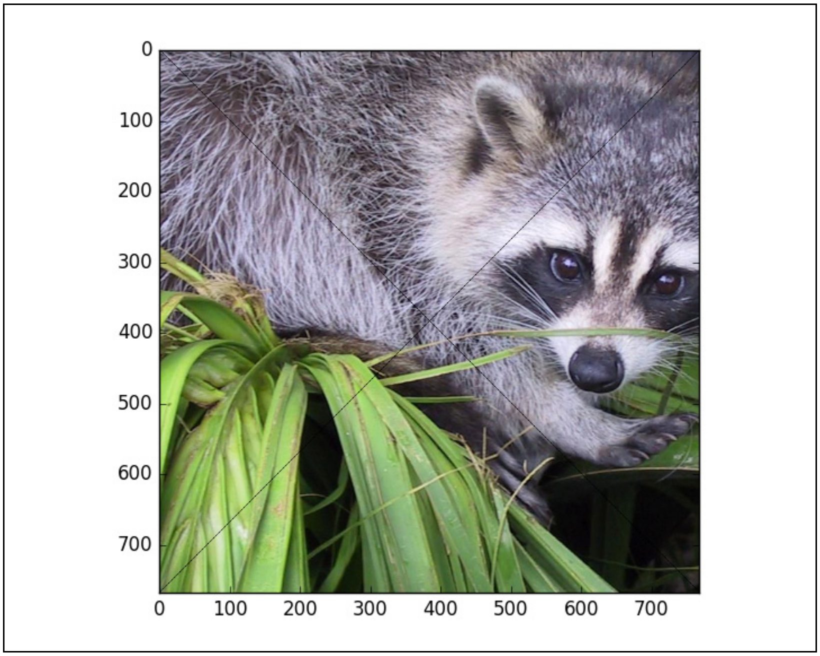


图 2-6

我们给 x 和 y 规定了不同的取值范围，这些范围用来索引莱娜数组。花式索引是在一个内部的 NumPy 迭代器对象的基础上实现的，分 3 步完成。

（1）创建迭代器对象。

（2）将迭代器对象绑定到数组。

（3）经由迭代器访问数组元素，利用位置列表进行索引。

2.9 基于位置列表的索引方法

下面利用 ix_()函数将莱娜照片中的像素完全打乱。注意，本例中的代码没有提供注释。完整的代码请参考本书代码包中的 ix.py 文件。

```
import scipy.misc
import matplotlib.pyplot as plt
import numpy as np

face = scipy.misc.face()
xmax = face.shape[0]
ymax = face.shape[1]

def shuffle_indices(size):
   arr = np.arange(size)
   np.random.shuffle(arr)

   return arr

xindices = shuffle_indices(xmax)
np.testing.assert_equal(len(xindices), xmax)
yindices = shuffle_indices(ymax)
np.testing.assert_equal(len(yindices), ymax)
plt.imshow(face[np.ix_(xindices, yindices)])
plt.show()
```

这个函数可以根据多个序列生成一个网格，它需要一个一维序列作为参数并返回一个由 NumPy 数组构成的元组。

```
In : ix_([0,1], [2,3])
Out:
(array([[0],[1]]), array([[2, 3]]))
```

利用位置列表索引 NumPy 数组的过程如下。

（1）打乱数组的索引。

我们用 numpy.random 子程序包中的 shuffle()函数把数组中的元素按随机的索引号重新排列，使得数组产生相应的变化。

```
def shuffle_indices(size):
   arr = np.arange(size)
```

```
    np.random.shuffle(arr)

return arr
```

（2）使用下面的代码画出打乱后的索引。

```
plt.imshow( face[np.ix_(xindices, yindices)])
```

（3）莱娜照片的像素被完全打乱后，变成图 2-7 所示的样子。

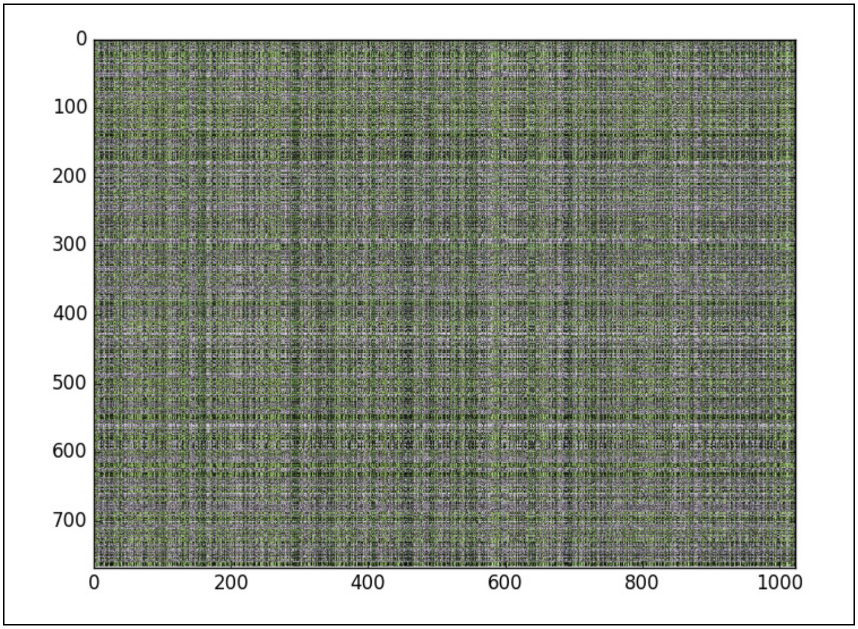


图 2-7

2.10 用布尔型变量索引 NumPy 数组

布尔型索引是指根据布尔型数组来索引元素的方法，属于花式索引系列。因为布尔型索引是花式索引的一个分类，所以它们的使用方法基本相同。

下面用代码（详见本书代码包中的 boolean_indexing.py 文件）具体演示其使用方法。

```
import scipy.misc
import matplotlib.pyplot as plt
import numpy as np

face = scipy.misc.face()
```

```
xmax = face.shape[0]
ymax = face.shape[1]
face=face[:min(xmax,ymax),:min(xmax,ymax)]

def get_indices(size):
   arr = np.arange(size)
   return arr % 4 == 0

face1 = face.copy()
xindices = get_indices(face.shape[0])
yindices = get_indices(face.shape[1])
face1[xindices, yindices] = 0
plt.subplot(211)
plt.imshow(face1)
face2 = face.copy()
face2[(face > face.max()/4) & (face < 3 * face.max()/4)] = 0
plt.subplot(212)
plt.imshow(face2)
plt.show()
```

上述代码利用一种特殊的迭代器对象来索引元素，下面进行简单说明。

（1）在对角线上画点。

这类似于花式索引，不过这里选择的是照片对角线上可以被 4 整除的那些位置上的点。

```
def get_indices(size):
 arr = np.arange(size)
 return arr % 4 == 0
```

然后我们仅绘出选定的那些点。

```
face1 = face.copy()
xindices = get_indices(face.shape[0])
yindices = get_indices(face.shape[1])
face1[xindices, yindices] = 0
plt.subplot(211)
plt.imshow(face1)
```

（2）根据元素值的情况置零。

选取数组值介于最大值的 1/4～3/4 的那些元素，将其设置为 0。

```
face2[(face > face.max()/4) & (face < 3 * face.max()/4)] = 0
```

（3）两幅新照片如图 2-8 所示。

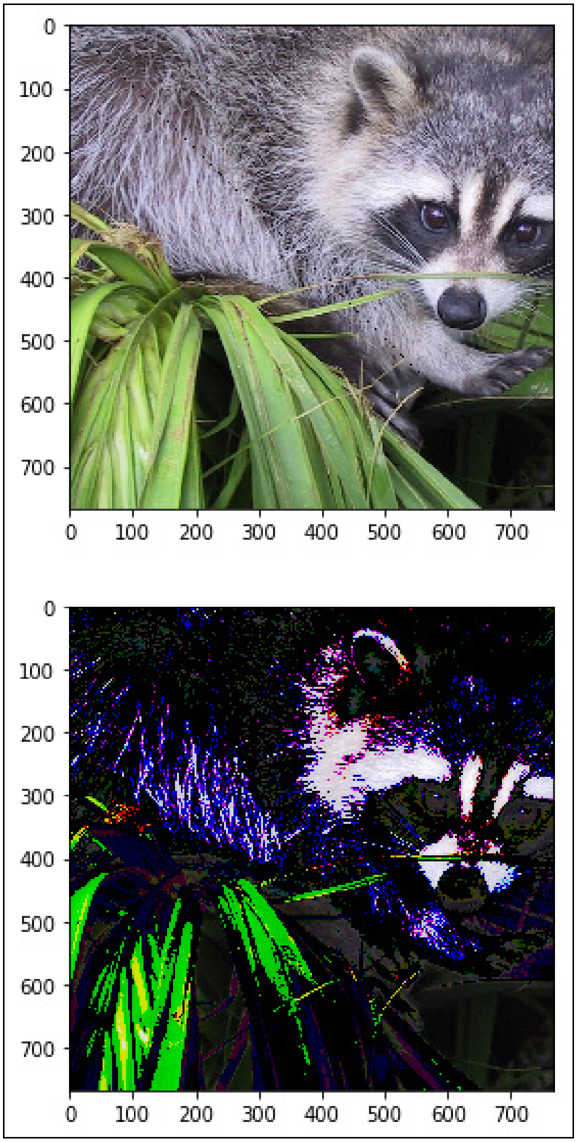


图 2-8

2.11　NumPy 数组的广播

当操作对象的形状不一样时，NumPy 会尽力进行处理。

例如，假设一个数组要跟一个标量相乘，这时标量需要根据数组的形状进行扩展，然后才可以执行乘法运算。这个扩展的过程叫作广播（broadcasting）。下面用代码加以说明。

```
import scipy.io.wavfile as sw
import matplotlib.pyplot as plt
import urllib
import numpy as np

request =
urllib.request.Request('http://www.thesoundarchive.com/austinpowers/smashin
gbaby.wav')
response = urllib.request.urlopen(request)
print(response.info())
WAV_FILE = 'smashingbaby.wav'
filehandle = open(WAV_FILE, 'wb')
filehandle.write(response.read())
filehandle.close()
sample_rate, data = sw.read(WAV_FILE)
print("Data type", data.dtype, "Shape", data.shape)

plt.subplot(2, 1, 1)
plt.title("Original")
plt.plot(data)

newdata = data * 0.2
newdata = newdata.astype(np.uint8)
print("Data type", newdata.dtype, "Shape", newdata.shape)

sw.write("quiet.wav", sample_rate, newdata)

plt.subplot(2, 1, 2)
plt.title("Quiet")
plt.plot(newdata)2

plt.show()
```

下面，我们将下载一个音频文件，然后以此为基础，生成一个新的静音版本。

（1）读取 WAV 文件。

我们将使用标准的 Python 代码来下载电影《王牌大贱谍》（Austin Powers）中的狂嚎式歌曲《Smashing, baby》的声音文件。SciPy 中有一个 wavfile 子程序包，可以用来加载音

频数据，或者生成 WAV 格式的文件。如果此前已经安装了 SciPy，那么现在就可以直接使用这个子程序包了。我们可以使用函数 read()读取文件，它返回一个数据阵列及采样率，不过，这里只对数据本身感兴趣。

```
sample_rate, data = scipy.io.wavfile.read(WAV_FILE)
```

（2）绘制原 WAV 数据。

这里，我们利用 matplotlib 绘制原始 WAV 数据并用一个子图来显示标题“Original”，代码如下。

```
plt.subplot(2, 1, 1)
plt.title("Original")
plt.plot(data)
```

（3）新建一个数组。

现在，我们要用 NumPy 来生成一段“寂静的”声音。实际上就是将原数组的值乘以一个常数，从而得到一个新数组，因为这个新数组的元素值肯定是变小了。这正是广播技术的用武之地。最后，我们要确保新数组与原数组的类型一致，即 WAV 格式。

```
newdata = data * 0.2
newdata = newdata.astype(np.uint8)
```

（4）写入一个 WAV 文件中。

我们将新数组保存到一个新的 WAV 文件中，代码如下。

```
scipy.io.wavfile.write("quiet.wav",
    sample_rate, newdata)
```

（5）绘制出新的 WAV 数据。

我们可以使用 matplotlib 来画出新数组中的数据，如下所示。

```
plt.subplot(2, 1, 2)
plt.title("Quiet")
plt.plot(newdata)
plt.show()
```

图 2-9 展示了原始的 WAV 文件中的数据的图像以及数值变小后新数组的图像。

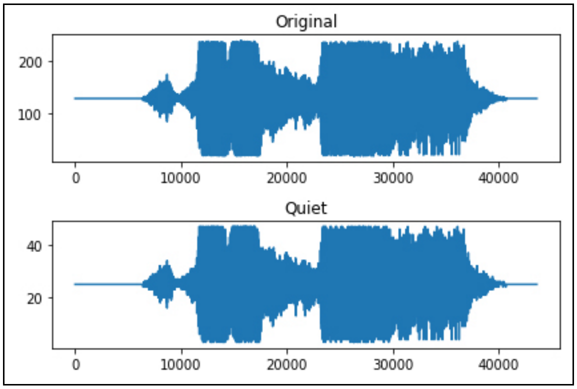


图 2-9

2.12 小结

本章，我们学习了 NumPy 的基础知识：数据类型和数组。数组具有许多属性，这些属性都是用来描述该数组的特性的。我们探讨的属性之一便是数据类型，实际上，NumPy 是通过一个成熟完备的对象来表示这个属性的。

与 Python 标准的列表相比，NumPy 数组使用的切片和索引方法更加高效。此外，NumPy 数组还能够对多维度数组进行处理。

我们可以用各种方式改变数组的形状，如堆叠、重定尺寸、重塑形状以及拆分等。在本章中，我们还为处理数组的形状介绍了许多简便易用的函数。

有了这些基础知识后，从第 4 章开始，我们就要学习如何通过常见的函数来分析数据了，这将涉及主要统计函数和数值函数的用法。

我们鼓励读者阅读“参考文献”部分中提到的书籍，以拓展大家在 NumPy 库知识方面的深度和广度。

2.13 参考资料

1. I. Idris, *NumPy Cookbook – Second edition*, Packt Publishing, 2015.

2. I. Idris, *Learning NumPy Array*, Packt Publishing, 2014.

3. I. Idris, *Numpy: Beginners Guide – Third Edition*, Packt Publishing, 2015.

4. L. (Liang-H.) Chin and T. Dutta, *NumPy Essentials*, Packt Publishing, 2016.

第 3 章 Pandas 入门

Pandas 是一个流行的开源 Python 程序库，其名称取 panel data（面板数据，一个计量经济学的术语）与 Python data analysis（Python 数据分析）之意。本章将向读者介绍 pandas 的基本功能，其中包括 Pandas 的数据结构与运算。

Pandas 的官方文档强调，Pandas 项目名称中的字母应该全部采用小写形式，同时还约定导入这个程序库时使用的语句为 import pandas as pd。

本书中，我们将尽可能地遵循这些惯例。

在本章中，我们首先安装并概要介绍 Pandas，然后开始探索 Pandas 的两个最重要的数据结构：DataFrame 和 Series。最后，我们将学习如何对存储在这些数据结构中的数据进行类似 SQL 这样的运算，还举例说明包括时间序列例程在内的统计学工具。本章涉及的主题如下。

- Pandas 的安装与概览
- Pandas 的数据结构：DataFrame
- Pandas 的数据结构：Series
- 利用 Pandas 查询数据
- 利用 Pandas 的 DataFrames 进行统计计算
- 利用 Pandas 的 DataFrames 聚合数据
- DataFrames 的连接（joining）与附加（appending）操作
- DataFrames 的串联（concatenating）操作
- 处理缺失数据问题

- 处理日期数据
- 数据透视表

3.1 Pandas 的安装与概览

对于 Pandas 来说，最小的依赖项集合如下。

- NumPy：这是一个处理数值数组的基础软件包，我们已经在前面的章节介绍过它的安装方法和简单用法。
- python-dateutil：这是一个专门用来处理日期数据的程序库。
- pytz：这是一个处理时区问题的程序库。

上面列出的是最低限度的依赖项，如果想更加全面地了解可选依赖项，请访问 http://pandas.pydata.org/pandas-docs/stable/install.html 页面。我们不仅能用 PyPI 的 pip 或者 easy_install 命令来安装 Pandas，而且可以使用已经编译好的二进制形式的安装程序。此外，我们还可以借助于操作系统的程序包管理器，甚至可以利用源代码来安装 Pandas。如果我们喜欢使用二进制形式的安装程序，可以从 Pandas 官网下载。

用 pip 程序安装 Pandas 的命令如下。

```
$ pip3 install pandas rpy2
```

rpy2 是 R 的一个接口，因为 rpy 被弃用，所以 rpy2 是必不可少的。如果我们的用户账号权限不够，则需要在相应的命令前面加上 sudo。

就像在第 1 章的 IPython Notebook 中看到那样，我们可以查看 Pandas 的版本及其子程序包。该程序将显示与 Pandas 有关的下列内容。

```
    pandas version 0.19.0pandas.apipandas.compat DESCRIPTION compat Crosscompatible
functions for Python 2 and 3. Key items to import for 2/3
    compatible code: * iterators: range(), map(),
    pandas.computationpandas.corepandas.formatspandas.indexespandas.iopandas.ms
    gpack DESCRIPTION # coding: utf-8 # flake8: noqa PACKAGE CONTENTS _packer
    _unpacker _version exceptions CLASSES ExtType(builtins.tuple) ExtType cl
    pandas.rpy DESCRIPTION # GH9602 # deprecate rpy to instead directly use
    rpy2 PACKAGE CONTENTS base common mass vars FILE
    /usr/local/lib/python3.5/sitepandas.sparsepandas.statspandas.testspandas.
toolspandas.tseriespandas.types
    pandas.util
```

遗憾的是，这里没有给出描述 Pandas 子库的详细信息，好在子程序包的名称本身就足以使人们对其有所了解。

3.2 Pandas 数据结构之 DataFrame

Pandas 的 DataFrame 数据结构是一种带标签的二维对象，与 Excel 的电子表格或者关系型数据库的数据表非常相似。顺便说一下，DataFrame 的概念最初发源于 R 程序语言。我们可以用下列方式来创建 DataFrame。

- 使用另一个 DataFrame 创建 DataFrame。
- 使用具有二维形状的 NumPy 数组或者数组的复合结构来生成 DataFrame。
- 类似地，可以用 Pandas 的另外一种数据结构 Series 来创建 DataFrame。关于 Series，后文将有所介绍。
- DataFrame 也可以从类似 CSV 之类的文件来生成。
- 使用一维 NumPy 数组、列表、字典或 Pandas Series 之类的一维结构的字典来生成 DataFrame。

下面以数据为例，来说明 Datarame。注意，原始的数据文件相当大，而且有很多列，所以我们将使用编辑后的仅保留前 9 列的一个文件作为替代。这个文件可以从本书的代码包中找到，文件的全称为 WHO_first9cols.csv。下面是该文件包括标题在内的前两行内容。

```
Country,CountryID,Continent,Adolescent fertility rate (%),Adult literacy rate (%),Gross national income per capita (PPP international $),Net primary school enrolment ratio female (%),Net primary school enrolment ratio male (%),Population (in thousands) totalAfghanistan,1,1,151,28,,,,26088
```

接下来，我们开始考察 Pandas 的 DataFrame 及其各种属性。

（1）首先，将数据文件载入 DataFrame 并显示其内容。

```
from pandas.io.parsers import read_csv

df = read_csv("WHO_first9cols.csv")
print("Dataframe", df)
```

这里显示的是该 DataFrame 的摘要信息。如果完整显示的话会很长，所以我们只是截

取了前面几行内容。

```
199                    21732.0
200                    11696.0
201                    13228.0

[202 rows x 9 columns]
```

（2）DataFrame 有一个属性，以元组的形式来存放 DataFrame 的形状数据，这与 ndarray 非常类似。我们可以查询一个 DataFrame 的行数，具体方法如下。

```
print("Shape", df.shape)
print("Length", len(df))
```

这里得到的数字与上一步的打印输出一致。

```
Shape (202, 9)
Length 202
```

（3）下面我们通过其他属性来考察各列的标题与数据类型，具体如下。

```
print("Column Headers", df.columns)
print("Data types", df.dtypes)
```

我们可以用一个专用的数据结构来容纳列标题（column headers）。

```
Column Headers Index([u'Country', u'CountryID', u'Continent',
u'Adolescent fertility rate (%)', u'Adult literacy rate (%)',
u'Gross national income per capita (PPP international $)',
u'Net primary school enrolment ratio female (%)',
u'Net primary school enrolment ratio male (%)',
u'Population (in thousands) total'], dtype='object')
```

该数据类型的输出结果如图 3-1 所示。

```
Data types Country                                                object
CountryID                                                  int64
Continent                                                  int64
Adolescent fertility rate (%)                            float64
Adult literacy rate (%)                                  float64
Gross national income per capita (PPP international $)   float64
Net primary school enrolment ratio female (%)            float64
Net primary school enrolment ratio male (%)              float64
Population (in thousands) total                          float64
```

图 3-1 输出结果

（4）Pandas 的 DataFrame 带有一个索引，类似于关系型数据库中数据表的主键（primary key）。对于这个索引，我们既可以手动规定，也可以让 pandas 自动创建。访问索引时，使用相应的属性即可，具体方法如下。

```
Print("Index", df.index)
```

利用索引可以迅速搜查数据项，正如书籍的索引一样。就本例而言，这个索引实际上是对数组的一种封装，它起始于 0，然后以 1 为单位逐行递增。

```
Index Int64Index([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12,
13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27,
28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42,
43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57,
58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72,
73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87,
88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, ...],
dtype='int64')
```

（5）有时我们希望遍历 DataFrame 的基础数据，如果使用 pandas 的迭代器，遍历列值的效率可能会很低。更好的解决方案是从基础的 NumPy 数组中提取这些数值，然后进行相应的处理。不过，pandas 的 DataFrame 提供了一个属性，可以在这方面为我们提供帮助。

```
print("Values", df.values)
```

注意，在输出中，一些非数字的数值被标为“nan”，这通常是由输入数据文件中的空字段引起的。

```
Values [['Afghanistan' 1 1 ..., nan nan 26088.0]
 ['Albania' 2 2 ..., 93.0 94.0 3172.0]
 ['Algeria' 3 3 ..., 94.0 96.0 33351.0]
 ...,
 ['Yemen' 200 1 ..., 65.0 85.0 21732.0]
 ['Zambia' 201 3 ..., 94.0 90.0 11696.0]
 ['Zimbabwe' 202 3 ..., 88.0 87.0 13228.0]]
```

上述展示的示例代码取自本书代码包中的 ch-03.ipynb 文件。

3.3　Pandas 数据结构之 Series

Pandas 的 Series 数据结构是由不同类型的元素组成的一维数组，该数据结构也具有标

签。我们可以通过下列方式来创建 Pandas 的 Series 数据结构。

- 使用 Python 的字典来创建 Series
- 使用 NumPy 数组来创建 Series
- 使用单个标量值来创建 Series

创建 Series 数据结构时，我们可以向构造函数递交一组轴标签，这些标签通常称为索引，是一个可选参数。默认情况下，如果使用 NumPy 数组作为输入数据，那么 Pandas 会将索引值从 0 开始自动递增。如果传递给构造函数的数据是一个 Python 字典，那么这个字典的键会经排序后变成相应的索引。如果输入数据是一个标量值，那么就需要由我们来提供相应的索引。索引中的每一个新值都要输入一个标量值。Pandas 的 Series 和 DataFrame 数据类型接口的特征和行为是从 NumPy 数组和 Python 字典那里借用来的，如切片、通过键查找以及向量化运算等。对一个 DataFrame 列执行查询操作时，会返回一个 Series 数据结构。关于这一点以及 Series 的其他特点，我们会以 3.2 节中的数据为例进行说明，所以需要再次加载 CSV 文件。

（1）首先，我们选中输入文件中的第一列，即 Country 列；然后，显示这个对象在局部作用域中的类型。

```
country_col = df["Country"]
print("Type df", type(df))
print("Type country col", type(country_col))
```

如今我们可以肯定，当选中 DataFrame 的一列时，得到的是一个 Series 型的数据，代码如下。

```
Type df <class 'pandas.core.frame.DataFrame'>
Type country col <class 'pandas.core.series.Series'>
```

> **提示**
> 如果需要，我们打开一个 Python 或者 IPython shell，导入 Pandas 后就可以通过 dir()函数来查看上述类型提供的各种函数和属性了，这时显示的函数列表将会更加全面。

（2）Pandas 的 Series 数据结构不仅共享了 DataFrame 的一些属性，还另外提供了与名称有关的一个属性。下面通过代码做进一步的探索。

```
print("Series shape", country_col.shape)
```

```
print("Series index", country_col.index)
print("Series values", country_col.values)
print("Series name", country_col.name)
```

输出内容（为了节约版面，这里仅截取了部分内容）如下。

```
Series shape (202,)
Series index Int64Index([0, 1, 2, 3, 4, 5,
6, 7, 8, 9, 10, 11, 12, ...], dtype='int64')
Series values ['Afghanistan' ... 'Vietnam' 'West Bank and
Gaza' 'Yemen' 'Zambia' 'Zimbabwe']
Series name Country
```

（3）为了演示 Series 的切片功能，这里以截取 Series 变量 Country 中的最后两个国家为例进行说明，具体代码如下。

```
print("Last 2 countries", country_col[-2:])
print("Last 2 countries type", type(country_col[-2:]))
```

就像我们看到的那样，利用切片技术，我们得到了一个新的 Series 数据对象。

```
Last 2 countries
200      Zambia
201    Zimbabwe
Name: Country, dtype: object
Last 2 countries type <class 'pandas.core.series.Series'>
```

（4）NumPy 的函数同样适用于 Pandas 的 DataFrame 和 Series 数据结构。例如，可以使用 NumPy 的 sign()函数来获得数字的符号：正数返回 1，负数返回−1，零值返回 0。下面将这个函数应用到前面的 DataFrame 数据及其最后一列上面，即数据集中各国的人口数量。

```
last_col = df.columns[-1]
print("Last df column signs:\n", last_col,
np.sign(df[last_col]), "\n")
```

为了节约版面，这里对输出内容进行了一定的删减，代码如下。

```
Last df column signs Population (in thousands) total 0     1
1     1
[TRUNCATED]
198   NaN
```

```
199     1
200     1
201     1
Name: Population (in thousands) total, Length: 202, dtype:
float64
```

> **提示**
>
> 注意，198 号索引对应的人口值为“NaN”，相应的数据项如下。
>
> ```
> West Bank and Gaza,199,1,,,,,,
> ```

我们可以在 DataFrames、Series 和 NumPy 数组之间进行各种类型的数值运算。举例来说，如果得到 pandas 的 Series 数据中的一个基础 NumPy 数组，然后把它从该 Series 中减掉，这时我们可能希望得到下面两种结果：

- 一个数组，各个元素以零填充，而且至少有一个 NaN（前面的步骤中已经提到过 NaN）；
- 一个元素全部为零值的数组。

对于 NumPy 函数来说，涉及 NaN 的大部分运算都会生成 NaN。下面我们使用一个 IPython 会话进行说明。

```
In: np.sum([0, np.nan])
Out: nan
```

利用下面的代码执行减法运算。

```
print np.sum(df[last_col] - df[last_col].values)
```

得到的数据与第二种类型的结果相符。

```
0.0
```

本节涉及的代码取自本书代码包中的 ch-03.ipynb 文件。

3.4 利用 Pandas 查询数据

因为 Pandas 的 DataFrame 的结构类似于关系型数据库，所以从 DataFrame 读取数据

可以看作是一种查询操作。下面的例子中，我们将会从 Quandl 检索年度太阳黑子数据。为此，我们既可以使用 Quandl API，也可以亲自动手从 http://www.quandl.com/SIDC/SUNSPOTS_A-Sunspot-Numbers-Annual 页面以 CSV 文件的形式下载数据。如果想安装 API，我们可以从 Quandl 网站下载相应的安装程序，或者运行以下命令。

```
$ pip3 install Quandl
```

> **提示**
> Quandl 的 API 可以免费使用，但是有一个限制，即每天最多调用 50 次。如果超过规定的调用次数，则需要申请身份验证密钥。不过，这里的代码无需使用密钥。如果需要使用密钥或读取下载的 CSV 文件，对这里的代码稍作修改即可。如果有困难，请参考第 1 章中从何处寻求帮助和参考资料部分，或者到官方指定的 Python 文档中查找解决方案。

话不多说，下面来看如何利用 Pandas 的 DataFrame 来查询数据。

（1）下载数据，导入 Quandl API 后，可以像下面这样来下载数据。

```
import quandl

# Data from
http://www.quandl.com/SIDC/SUNSPOTS_A-Sunspot-Numbers-Annual
# PyPi url https://pypi.python.org/pypi/Quandl
sunspots = quandl.get("SIDC/SUNSPOTS_A")
```

（2）head()和 tail()这两个函数的作用类似于 UNIX 系统中同名的两个命令，即选取 DataFrame 的前 n 个和后 n 个数据记录，其中 n 是一个整型参数。

```
print("Head 2", sunspots.head(2) )
print("Tail 2", sunspots.tail(2))
```

下面给出太阳黑子数据的前两条和后两条数据的内容（为了保持版面简洁起见，没有在这里显示所有列；不过在运行代码的时候，输出中将含有数据集的所有列）。

```
Head 2          Number
Year
1700-12-31      5
```

```
1701-12-31        11

[2 rows x 1 columns]
Tail 2             Number
Year
2012-12-31    57.7
2013-12-31    64.9
[2 rows x 1 columns]
```

注意，我们有且只有一列数据来存放每年太阳黑子的数量，而日期属于 DataFrame 索引的一部分。

（3）下面用最近的日期来查询最近一年太阳黑子的相关数据。

```
last_date = sunspots.index[-1]
print("Last value", sunspots.loc[last_date])
```

我们可以将下面的输出内容与前面得到的结果进行比对。

```
Last value Number 64.9
Name: 2013-12-31 00:00:00, dtype: float64
```

（4）下面介绍如何通过 YYYYMMDD 格式的日期字符串来查询日期，具体如下。

```
print("Values slice by date:\n", sunspots["20020101":
"20131231"])
```

以上结果为 2002－2013 年的数据记录。

```
Values slice by date                Number
Year
2002-12-31   104.0
[TRUNCATED]
2013-12-31    64.9
[12 rows x 1 columns]
```

（5）索引列表也可用于查询，代码如下。

```
print("Slice from a list of indices:\n", sunspots.iloc[[2, 4,
-4, -2]])
```

上述代码会选择下列各行。

```
Slice from a list of indices        Number
```

```
Year
1702-12-31    16.0
1704-12-31    36.0
2010-12-31    16.0
2012-12-31    57.7

[4 rows x 1 columns]
```

（6）要想选择标量值，有两种方法，这里给出的是速度明显占优势的第二种方法。它需要两个整数作为参数，其中第一个整数表示行，第二个整数表示列。

```
print("Scalar with Iloc:", sunspots.iloc[0, 0])
print("Scalar with iat", sunspots.iat[1, 0])
```

上面的代码将数据集的第一个值和第二个值作为标量返回。

```
Scalar with Iloc 5.0
Scalar with iat 11.0
```

（7）查询布尔型变量的方法与 SQL 的 Where 子句非常接近。下面的代码将查询大于算术平均值的各个数值。注意，在整个 DataFrame 中进行查询与在单列上进行查询是有区别的。

```
print("Boolean selection", sunspots[sunspots >
sunspots.mean()])
print("Boolean selection with column label:\n",
sunspots[sunspots['Number of Observations'] > sunspots['Number
of Observations'].mean()])
```

显著的区别在于，第一个查询操作得到的是所有数据行，其中与条件不符的行将被赋予 NaN 值，第二个查询操作返回的只是其值大于平均值的那些行。

```
Boolean selection                  Number
Year
1700-12-31    NaN
[TRUNCATED]
1759-12-31    54.0
                ...

[314 rows x 1 columns]
Boolean selection with column label                Number
Year
1705-12-31    58.0
```

```
[TRUNCATED]
1870-12-31    139.1
                ...

[127 rows x 1 columns]
```

以上示例代码取自本书代码包中的 ch_03.ipynb 文件。

3.5 利用 Pandas 的 DataFrame 进行统计计算

Pandas 的 DataFrame 数据结构为我们提供了若干统计函数，表 3-1 给出了部分方法及其简要说明。

表 3-1

方法	说明
describe	这个方法将返回描述性统计信息
count	这个方法将返回非 NaN 数据项的数量
mad	这个方法用于计算平均绝对偏差（mean absolute deviation），即类似于标准差的一个有力统计工具
median	这个方法将返回中位数，等价于第 50 位百分位数的值
min	这个方法将返回最小值
max	这个方法将返回最大值
mode	这个方法将返回众数（mode），即一组数据中出现次数最多的变量值
std	这个方法将返回表示离散度（dispersion）的标准差，即方差的平方根
var	这个方法将返回方差
skew	这个方法用来返回偏态系数（skewness），该系数表示的是数据分布的对称程度
kurt	这个方法将返回峰态系数（kurtosis），该系数用来反映数据分布曲线顶端尖峭或扁平程度

下面使用上例中的数据来演示这些统计函数的使用方法。如果读者对完整的代码感兴趣，请参考本书代码包中的 ch-03.ipynb 文件。演示代码如下。

```
import quandl
```

```
# Data from http://www.quandl.com/SIDC/SUNSPOTS_A-Sunspot-Numbers-Annual
# PyPi url https://pypi.python.org/pypi/Quandl
sunspots = quandl.get("SIDC/SUNSPOTS_A")
print("Describe", sunspots.describe(),"\n")
print("Non NaN observations", sunspots.count(),"\n")
print("MAD", sunspots.mad(),"\n")
print("Median", sunspots.median(),"\n")
print("Min", sunspots.min(),"\n")
print("Max", sunspots.max(),"\n")
print("Mode", sunspots.mode(),"\n")
print("Standard Deviation", sunspots.std(),"\n")
print("Variance", sunspots.var(),"\n")
print("Skewness", sunspots.skew(),"\n")
print("Kurtosis", sunspots.kurt(),"\n")
```

以上脚本的运行结果如图 3-2、图 3-3 所示。

```
Describe        Yearly Mean Total Sunspot Number  Yearly Mean Standard Deviation  \
count                         316.000000                      198.000000
mean                           79.503481                        8.030303
std                            62.057114                        3.807299
min                             0.000000                        1.700000
25%                            25.050000                        4.725000
50%                            66.700000                        7.700000
75%                           116.400000                       10.475000
max                           269.300000                       19.100000

       Number of Observations  Definitive/Provisional Indicator
count              198.000000                             316.0
mean              1424.888889                               1.0
std               2394.898980                               0.0
min                150.000000                               1.0
25%                365.000000                               1.0
50%                365.000000                               1.0
75%                366.000000                               1.0
max               8903.000000                               1.0

Non NaN observations Yearly Mean Total Sunspot Number    316
Yearly Mean Standard Deviation      198
Number of Observations              198
Definitive/Provisional Indicator    316
dtype: int64

MAD Yearly Mean Total Sunspot Number      50.987620
Yearly Mean Standard Deviation         3.125375
Number of Observations              1777.463524
Definitive/Provisional Indicator       0.000000
dtype: float64

Median Yearly Mean Total Sunspot Number     66.7
Yearly Mean Standard Deviation       7.7
Number of Observations              365.0
Definitive/Provisional Indicator      1.0
dtype: float64
```

图 3-2

```
Min Yearly Mean Total Sunspot Number      0.0
Yearly Mean Standard Deviation        1.7
Number of Observations              150.0
Definitive/Provisional Indicator      1.0
dtype: float64

Max Yearly Mean Total Sunspot Number     269.3
Yearly Mean Standard Deviation         19.1
Number of Observations               8903.0
Definitive/Provisional Indicator       1.0
dtype: float64

Mode    Yearly Mean Total Sunspot Number  Yearly Mean Standard Deviation  \
0                              18.3                             9.2

   Number of Observations  Definitive/Provisional Indicator
0                   365.0                               1.0

Standard Deviation Yearly Mean Total Sunspot Number      62.057114
Yearly Mean Standard Deviation         3.807299
Number of Observations              2394.898980
Definitive/Provisional Indicator       0.000000
dtype: float64

Variance Yearly Mean Total Sunspot Number    3.851085e+03
Yearly Mean Standard Deviation      1.449552e+01
Number of Observations              5.735541e+06
Definitive/Provisional Indicator    0.000000e+00
dtype: float64

Skewness Yearly Mean Total Sunspot Number    0.799452
Yearly Mean Standard Deviation      0.555067
Number of Observations              1.876098
Definitive/Provisional Indicator    0.000000
dtype: float64

Kurtosis Yearly Mean Total Sunspot Number   -0.143733
Yearly Mean Standard Deviation     -0.244310
Number of Observations              1.783261
Definitive/Provisional Indicator    0.000000
dtype: float64
```

图 3-3 脚本运行结果

3.6 利用 Pandas 的 DataFrame 实现数据聚合

数据聚合（data aggregation）是关系型数据库中比较常用的一个术语。使用数据库时，我们可以利用查询操作对各列或各行中的数据进行分组，这样就可以针对其中的每一组数据进行各种不同的操作了。实际上，Pandas 的 DataFrame 数据结构也为我们提供了类似的

功能。我们可以把生成的数据保存到 Python 字典中，然后利用这些数据来创建一个 Pandas DataFrame，接下来就可以练习 Pandas 提供的聚合功能了。

（1）我们为 NumPy 的随机数生成器指定种子，确保重复运行程序时生成的数据不会走样。该数据有 4 列，分别是：

- Weather（一个字符串）
- Food（一个字符串）
- Price（一个随机浮点数）
- Number（1～9 之间的一个随机整数）

假设有一些客户消费调查、天气与市场定价方面的资料，这里要做的是计算平均价格并跟踪样本的大小及参数。

```
import pandas as pd
from numpy.random import seed
from numpy.random import rand
from numpy.random import rand_int
import numpy as np

seed(42)

df = pd.DataFrame({'Weather' : ['cold', 'hot', 'cold',
'hot', 'cold', 'hot', 'cold'],
'Food' : ['soup', 'soup', 'icecream', 'chocolate',
'icecream', 'icecream', 'soup'],
'Price' : 10 * rand(7), 'Number' : rand_int(1, 9,)})
print(df)
```

我们将得到下面的结果。

```
       Food Number     Price Weather
0        soup      8  3.745401    cold
1        soup      5  9.507143     hot
2    icecream      4  7.319939    cold
3   chocolate      8  5.986585     hot
4    icecream      8  1.560186    cold
5    icecream      3  1.559945     hot
6       soup       6  0.580836    cold

[7 rows x 4 columns]
```

> **提示**
>
> 注意，列标签是按照 Python 字典各个键的词汇顺序进行排列的。所谓的词汇或者词典顺序，就是按照字符串中各个字符的字母表顺序进行排序。

（2）通过 Weather 列为数据分组，然后遍历各组数据，代码如下。

```
weather_group = df.groupby('Weather')

i = 0
for name, group in weather_group:
   i = i + 1
   print("Group", i, name)
   print(group)
```

我们把天气状况分为两种，即热天与冷天，据此可以把数据分为两组。

```
Group 1 cold
       Food  Number     Price Weather
0      soup       8  3.745401    cold
2  icecream       4  7.319939    cold
4  icecream       8  1.560186    cold
6      soup       6  0.580836    cold

[4 rows x 4 columns]
Group 2 hot
        Food  Number     Price Weather
1       soup       5  9.507143     hot
3  chocolate       8  5.986585     hot
5   icecream       3  1.559945     hot

[3 rows x 4 columns]
```

（3）变量 Weather_group 是一种特殊的 Pandas 对象，可由 groupby()生成。这个对象为我们提供了聚合函数，下面我们展示它的使用方法。

```
print("Weather group first\n", weather_group.first())
print("Weather group last\n", weather_group.last())
print("Weather group mean\n", weather_group.mean())
```

以上代码将输出各组数据的第 1 行内容、第 2 行内容以及各组的平均值，显示如下。

```
Weather group first
         Food  Number      Price
Weather
```

```
cold     soup       8   3.745401
hot      soup       5   9.507143

[2 rows x 3 columns]
Weather group last
         Food  Number      Price
Weather
cold         soup       6   0.580836
hot      icecream       3   1.559945

[2 rows x 3 columns]
Weather group mean          Number   Price
Weather
cold     6.500000  3.301591
hot      5.333333  5.684558

[2 rows x 2 columns]
```

（4）恰如利用数据库的查询操作那样，我们也可以针对多列进行分组。

此后，我们便可以利用 groups 属性来了解所生成的数据组以及每一组包含的行数。

```
wf_group = df.groupby(['Weather', 'Food'])
print("WF Groups", wf_group.groups)
```

针对天气数据和食物数据的每一种可能的组合，都会为其生成一个新的数据组。每一行中的各个数据项都可以通过索引值引用，具体如下。

```
WF Groups {('hot', 'chocolate'): [3], ('cold', 'icecream'):
[2, 4], ('hot', 'icecream'): [5], ('hot', 'soup'): [1],
('cold', 'soup'): [0, 6]}
```

（5）通过 agg()方法，可以对数据组施加一系列的 NumPy 函数。

```
print("WF Aggregated\n", wf_group.agg([np.mean, np.median]))
```

显而易见，我们也可以施加更多的函数。不过这样做的结果是，它的输出结果会比下面的输出结果更乱。

```
WF Aggregated                    Number             Price
                                   mean  median       mean    median
Weather Food
cold    icecream                      6       6   4.440063  4.440063
        soup                          7       7   2.163119  2.163119
hot     chocolate                     8       8   5.986585  5.986585
        icecream                      3       3   1.559945  1.559945
```

```
          soup                               5        5  9.507143  9.507143

[5 rows x 4 columns]
```

完整的数据聚合示例代码请参考 data_aggregation.py 文件，这个文件位于本书的代码包中。

3.7 DataFrame 的串联与附加操作

数据库的数据表有内部连接和外部连接两种连接操作类型。实际上，Pandas 的 DataFrame 也有类似的操作，因此我们也可以对数据行进行串联和附加。我们将使用前面章节中的 DataFrame 来练习数据行的串联与附加操作。我们首先选中前面 3 行数据。

```
print("df :3\n", df[:3])
```

下面我们来看输出的是否为前 3 行的内容。

```
df :3
       Food  Number     Price Weather
0      soup       8  3.745401    cold
1      soup       5  9.507143     hot
2  icecream       4  7.319939    cold
```

函数 concat()的作用是串联 DataFrame，例如可以把一个由 3 行数据组成的 DataFrame 与其他数据行串接，以便重建原 DataFrame。

```
print("Concat Back together\n", pd.concat([df[:3], df[3:]]))
```

串联后的效果如下。

```
Concat Back together
        Food  Number     Price Weather
0       soup       8  3.745401    cold
1       soup       5  9.507143     hot
2   icecream       4  7.319939    cold
3  chocolate       8  5.986585     hot
4   icecream       8  1.560186    cold
5   icecream       3  1.559945     hot
6       soup       6  0.580836    cold

[7 rows x 4 columns]
```

为了追加数据行，可以使用 append()函数。

```
print("Appending rows\n", df[:3].append(df[5:]))
```

我们得到的 DataFrame 的前 3 行数据来自原 DataFrame，后面两行数据是附加上的。

```
Appending rows
       Food  Number     Price Weather
0      soup       8  3.745401    cold
1      soup       5  9.507143     hot
2  icecream       4  7.319939    cold
5  icecream       3  1.559945     hot
6      soup       6  0.580836    cold

[5 rows x 4 columns]
```

3.8 连接 DataFrames

为演示连接操作，我们要用到两个 CSV 文件，dest.csv 与 tips.csv 文件。假设我们正在运营一家出租车公司，每当乘客在目的地下车后，我们就需要向 dest.csv 文件增加一行数据，内容为出租车司机的员工编号与目的地。

```
EmpNr,Dest5,The Hague3,Amsterdam9,Rotterdam
```

有时，出租车司机会收到一些小费，因此我们希望把它登记到 tips.csv 文件中（这听起来好像不太现实，如果读者有好的案例，请随时提供给我们）。

```
EmpNr,Amount5,109,57,2.5
```

Pandas 提供的 merge()函数或 DataFrame 的 join()实例方法都能实现类似数据库的连接操作功能。默认情况下，join()实例方法会按照索引进行连接。不过，有时候这不符合我们的要求。使用关系型数据库查询语言（SQL）时，可以进行内部连接、左外连接、右外连接与完全外部连接等操作。

提示

对于内部连接，它会从两个数据表中选取数据，只要两个表中连接条件规定的列上存在相匹配的值，相应的数据就会被组合起来。对于外部连接，因为不要求进行匹配处理，所以将返回更多的数据。关于连接操作的进一步介绍，读者可以参考维基百科页面。

虽然 Pandas 支持所有的这些连接类型，限于篇幅，这里只介绍内部连接与完全外部连接。

- 我们用 merge()函数按照员工编号进行连接处理，代码如下。

```
print("Merge() on key\n", pd.merge(dests, tips, on='EmpNr'))
```

下面给出内部连接的结果。

```
Merge() on key
   EmpNr        Dest   Amount
0      5   The Hague       10
1      9   Rotterdam        5

[2 rows x 3 columns]
```

- 用 join()方法执行连接操作时，需要使用后缀来指示左操作对象和右操作对象。

```
print("Dests join() tips\n", dests.join(tips, lsuffix='Dest',
rsuffix='Tips'))
```

这个方法会连接索引值，因此得到的结果与 SQL 内部连接会有所不同。

```
Dests join() tips
   EmpNrDest        Dest   EmpNrTips  Amount
0          5   The Hague           5    10.0
1          3   Amsterdam           9     5.0
2          9   Rotterdam           7     2.5

[3 rows x 4 columns]
```

- 用 merge()执行内部连接时，更显式的方法如下。

```
print("Inner join with merge()\n", pd.merge(dests, tips,
how='inner'))
```

输出内容如下。

```
Inner join with merge()
   EmpNr        Dest  Amount
0      5   The Hague      10
1      9   Rotterdam       5

[2 rows x 3 columns]
```

只要稍作修改，就可以变成完全外部连接。

```
print("Outer join\n", pd.merge(dests, tips, how='outer'))
```

此外部连接操作增添了几行带有 NaN 值的数据。

```
Outer join
   EmpNr        Dest  Amount
0      5  The Hague    10.0
1      3  Amsterdam     NaN
2      9  Rotterdam     5.0
3      7        NaN     2.5

[4 rows x 3 columns]
```

如果使用关系型数据库的查询操作，这些数据都会被设为 NULL。以上演示代码取自本书代码包中的 ch - 03.ipynb 文件：

3.9 处理缺失数据问题

无论我们喜不喜欢，都会经常在数据记录中遇到空字段，所以我们最好接受这个现实并且学习如何通过一种可行的方式来解决这种问题。现实中的数据不仅有遗漏的现象，而且其中的某些数据还可能是错误的，因为所有的测量设备都难免会出现故障。对于 Pandas 来说，它会把缺失的数值标为 NaN，表示 None；还有一个类似的符号是 NaT，不过，它代表的是 datetime64 型对象。对 NaN 这个数值进行算术运算时，得到的结果还是 NaN。如果描述性统计学方法遇到这种值，如求和与求均值时，结果就不同了。就像我们在前面的例子中看到的那样，有时 NaN 被当成零值来进行处理。然而，在诸如求和之类的运算过程中，如果所有的数值都是 NaN，返回的结果仍然是 NaN。进行聚合操作时，我们组合的列内的 NaN 值会被忽略。下面我们会重新把 WHO_first9cols.csv 加载到一个 DataFrame 中。注意，这个文件中含有许多空白字段。这里只选取前 3 行数据，其中包括 Country 列与 Net primary school enrolment ratio male (%)列的标题，具体代码如下。

```
df = df[['Country', df.columns[-2]]][:2]
print("New df\n", df)
```

这样，我们就得到一个包含两个 NaN 值的 DataFrame 数据结构，具体内容如下。

```
New df
       Country  Net primary school enrolment ratio male (%)
```

```
0  Afghanistan                                          NaN
1      Albania                                           94

[2 rows x 2 columns]
```

Pandas 的 isnull()函数可以帮我们检查缺失的数据，使用方法如下。

```
print("Null Values\n", pd.isnull(df))
```

对于我们的 DataFrame，它的输出如下。

```
Null Values
  Country Net primary school enrolment ratio male (%)
0   False                                         True
1   False                                        False
```

如果要统计每一列中 NaN 值的数量，只要对 isnull()函数返回的布尔值进行求和即可。这种方法之所以奏效，是因为在求和的过程中 True 值被视为 1，而 False 值被视为 0 来对待。

```
Total Null Values
Country                                         0
Net primary school enrolment ratio male (%)    1
dtype: int64
```

类似地，可以用 DataFrame 的 notnull()方法来考察非缺失数据。

```
print("Not Null Values\n", df.notnull())
```

notnull()方法的返回结果与 isnull()函数的返回结果正好相反。

```
Not Null Values
  Country Net primary school enrolment ratio male (%)
0    True                                        False
1    True                                         True
```

如果一个 DataFrame 含有 NaN 值，那么将这个 DataFrame 中的值都乘 2 后，结果还是含有 NaN 值，因为“乘 2”是一种算术运算。

```
print("Last Column Doubled\n", 2 * df[df.columns[-1]])
```

下面用 2 乘以含有数值的最后一列（字符串乘以 2 表示重复该字符串）。

```
Last Column Doubled
0    NaN
```

```
1    188
Name: Net primary school enrolment ratio male (%), dtype: float64
```

然而，如果加一个 NaN 值，NaN 值就会大获全胜。

```
print("Last Column plus NaN\n", df[df.columns[-1]] + np.nan)
```

如你所见，现在变成 NaN 值的天下了。

```
Last Column plus NaN
0   NaN1 NaN
Name: Net primary school enrolment ratio male (%), dtype: float64
```

通过 fillna()方法，可以用一个标量值（如 0）来替换缺失数据，尽管有时可以用 0 替换缺失数据，但是事情并不总是如此。

```
print("Zero filled\n", df.fillna(0))
```

执行上面的代码后，NaN 值就会被 0 替换。

```
Zero filled
       Country Net primary school enrolment ratio male (%)
0  Afghanistan                                           0
1      Albania                                          94
```

本节代码选自本书代码包中的 ch-03.ipynb 文件。

3.10　处理日期数据

日期数据处理起来比较复杂，如 Y2K 问题、悬而未决的 2038 年问题以及时区问题等。虽然日期数据的处理很棘手，但是在处理时间序列数据时，这种数据是必不可少的。好在 Pandas 可以帮我们确定日期区间、对时间序列数据重新采样以及对日期进行算术运算。

下面，我们设定一个自 1900 年 1 月 1 日开始为期 42 天的时间范围，具体如下。

```
print("Date range", pd.date_range('1/1/1900', periods=42, freq='D'))
```

当然，1 月肯定不足 42 天，因此结束日期位于 2 月，不信可以检查一下。

```
Date range <class 'pandas.tseries.index.DatetimeIndex'>
[1900-01-01, ..., 1900-02-11]
Length: 42, Freq: D, Timezone: None
```

表 3-2 引用自 Pandas 的官方文档，描述了 Pandas 用的各种频率。

表 3-2

短码	说明
B	营业日频率
C	自定义营业日频率（试验性的）
D	日历日频率
W	周频率
M	月末频率
BM	营业月末频率
MS	月初频率
BMS	营业月初频率
Q	季末频率
BQ	营业季末频率
QS	季初频率
BQS	营业季初频率
A	年终频率
BA	营业年终频率
AS	年初频率
BAS	营业年初频率
H	小时频率
T	分钟频率
S	秒频率
L	毫秒
U	微秒

在 Pandas 中，日期区间是有限制的。Pandas 的时间戳基于 NumPy datetime64 类型，以纳秒（十亿分之一秒）为单位，而且用一个 64 位整数来表示具体数值。因此，日期有效的时间戳大体介于 1677—2262 年这段时间范围内。当然，这些年份中也不是所有

日期都是有效的。这个时间范围的精确中点应该是 1970 年 1 月 1 日。这样，1677 年 1 月 1 日就无法用 Pandas 的时间戳定义，而 1677 年 9 月 30 日则可以，下面我们用代码进行说明。

```
try:
   print("Date range", pd.date_range('1/1/1677', periods=4, freq='D'))
except:
   etype, value, _ = sys.exc_info()
   print("Error encountered", etype, value)
```

以上代码会得到如下错误信息。

```
Date range Error encountered <class 'pandas.tslib.OutOfBoundsDatetime'> Out
of bounds nanosecond timestamp: 1677-01-01 00:00:00
```

根据前面的介绍，下面用 Pandas 的 DateOffset 函数来计算允许的日期范围，代码如下。

```
offset = DateOffset(seconds=2 ** 33/10 ** 9)
mid = pd.to_datetime('1/1/1970')
print("Start valid range", mid - offset)
print("End valid range", mid + offset')
```

结果如下。

```
Start valid range 1969-12-31 23:59:51.410065
End valid range 1970-01-01 00:00:08.589935
```

我们可以用 Pandas 把一列字符串转换成日期数据，当然，并非所有的字符串都可以进行转换。当 Pandas 无法对一个字符串进行转换时，就会报错。由于不同地区对日期的定义方式会有所不同，这时就会产生歧义。下面以格式串为例进行说明，具体如下。

```
print("With format", pd.to_datetime(['19021112', '19031230'],
format='%Y%m%d'))
```

这两个字符串可以正确地进行转换，所以不会报错。

```
With format [datetime.datetime(1902, 11, 12, 0, 0) datetime.datetime(1903,
12, 30, 0, 0)]
```

如果一个字符串明显不是日期，那么默认情况下是无法进行转换的。

```
print("Illegal date", pd.to_datetime(['1902-11-12', 'not a date']))
```

很明显，上面的第 2 个字符串是无法进行转换的。

```
Illegal date ['1902-11-12' 'not a date']
```

如果需要进行强制转换，必须把参数 coerce 设置为 True。

```
print("Illegal date coerced", pd.to_datetime(['1902-11-12', 'not a date'],
errors='coerce'))
```

显而易见，第 2 个字符串仍然无法转化为一个日期，所以最终得到的是一个非时间数据 NaT。

```
Illegal date coerced <class 'pandas.tseries.index.DatetimeIndex'>
[1902-11-12, NaT]Length: 2, Freq: None, Timezone: None
```

该例子的示例代码取自本书代码包中的 ch-03.ipynb 文件。

3.11 数据透视表

熟悉 Excel 的读者都知道，数据透视表可以用来汇总数据。目前为止，本章中所见的 CSV 文件中的数据都是以平面文件的形式存放的。数据透视表可以从一个平面文件中指定的行和列中聚合数据，这种聚合操作可以是求和、求平均值、求标准差等运算。这里我们将再次用到前面 data_aggregation.py 的相关数据。因为 Pandas API 已经为我们提供了顶级 pivot_table()函数以及相应的 DataFrame 方法，所以，只要设置好 aggfunc 参数，就可以让这个聚合函数来执行 NumPy 中诸如 sum()之类的函数。参数 cols 用来告诉 Pandas 要对哪些列进行聚合运算。下面针对 Food 列来创建一个数据透视表，具体如下。

```
print(pd.pivot_table(df, cols=['Food'], aggfunc=np.sum))
```

我们得到的数据透视表包含了完整的食物数据项。

```
Food    chocolate   icecream      soup
Number   8.000000  15.000000  19.00000
Price    5.986585  10.440071  13.83338

[2 rows x 3 columns]
```

以上代码取自本书代码包中的 ch-03.ipynb 文件。

3.12 小结

本章主要介绍了Pandas，它是Python的数据分析程序库。我们可以将本章看成是Pandas功能特性与数据结构的基础入门资料。本章介绍了 Pandas 各种类似关系型数据库数据表的功能，通过这些功能，我们可以高效地进行查询、聚合及其他数据操作。通过配合使用NumPy 和 Pandas，我们可以完成各种基本的统计分析任务。现在，读者是不是觉得对于数据分析来说，有了 Pandas 就万事俱备了呢？实际上，这还远远不够。

掌握了本章介绍的基本之后，我们将在第 4 章中详解数据分析过程中常用的统计和数值函数。

我们鼓励读者阅读参考文献中介绍的各种书籍，以便更加详细和深入地探索 Pandas。

3.13 参考资料

1. Ted Petrou, *Pandas Cookbook*, Packt Publishing, 2017.

2. F. Anthony, *Mastering pandas*, Packt Publishing, 2015.

3. M. Heydt, *Mastering pandas for Finance*, Packt Publishing, 2015.

4. T. Hauck, *Data Intensive Apps with pandas How-to*, Packt Publishing, 2013.

5. M. Heydt, *Learning pandas*, Packt Publishing, 2015.

第 4 章
统计学与线性代数

统计学和线性代数为探索性数据分析奠定了基础。无论是描述性还是推论性的统计方法，都有助于从原始数据中获得见解和进行推断。比如，通过数据求出变量的算术平均值和标准差，并由此推出该变量的取值范围和期望值后，我们就可以利用统计检验来评估所得结论的可信度了。

线性代数关注的是解线性方程组，而 NumPy 和 SciPy 的 linalg 程序包可以帮我们轻松地解决这个问题。线性代数用途广泛，如利用模型拟合数据时就离不开它。除此之外，本章还会介绍其他几种 NumPy 和 SciPy 程序包，内容涉及随机数的生成和掩码式数组（Masked arrays）。

本章涉及以下主题。

- 利用 NumPy 处理简单的描述性统计运算
- 利用 NumPy 进行线性代数运算
- 利用 NumPy 寻找特征值和特征向量
- NumPy 随机数
- 创建 NumPy 掩码式数组（Masked arrays）

4.1 用 NumPy 进行简单的描述性统计计算

在本书中，我们会尽量使用各种不同的可以通过公开渠道获得的数据集。但是，这些数据的主题未必正是读者的兴趣之所在。此外，虽然每个数据集都有其自身的特点，但是本书介绍的技巧却是通用的，所以同样适用于大家的领域。在本章中，我们将数据集从

statsmodels 库载入 NumPy 数组，来进行数据分析。

Mauna Loa CO_2 测量数据是我们用到的 statsmodels 库的第一个数据集。以下代码将会加载数据集并给出基本的描述性统计值。

```
import numpy as np
from scipy.stats import scoreatpercentile
import pandas as pd

data = pd.read_csv("co2.csv", index_col=0, parse_dates=True)
co2 = np.array(data.co2)

print("The statistical values for amounts of co2 in atmosphere: \n")
print("Max method", co2.max())
print("Max function", np.max(co2))

print("Min method", co2.min())
print("Min function", np.min(co2))

print("Mean method", co2.mean())
print("Mean function", np.mean(co2))

print("Std method", co2.std())
print("Std function", np.std(co2))

print("Median", np.median(co2))
print("Score at percentile 50", scoreatpercentile(co2, 50))
```

上面的代码会计算 NumPy 数组的平均值、中位数、最大值、最小值以及标准差。

> **提示**
> 如果这些术语听起来不太熟悉，不妨花些时间到维基百科或其他地方了解一下相关内容。前言部分讲过，本书假设读者熟悉基本的数学和统计概念。

这里使用的数据来自 statsmodels 包，这些都是美国夏威夷 Mauna Loa 天文台观测到的大气 CO_2 的值。

现在，我们来看看具体代码。

（1）首先，使用 import 语句来加载 Python 包的相关模块，具体如下。

```
import numpy as np
from scipy.stats import scoreatpercentile
```

```
import pandas as pd
```

（2）然后，调用下面的函数来加载数据。

```
data = pd.read_csv("co2.csv", index_col=0, parse_dates=True)
co2 = np.array(data.co2)
```

上述代码中的数据将被复制到 NumPy 数组 co2 中，该数组用来存放大气中 CO_2 的观测值。

（3）数组的最大值可以通过 ndarray 的方法或者其他的 NumPy 函数来获得，另外最小值、平均值和标准差同样如此。下列代码将会显示各种统计指标。

```
print("Max method", co2.max())
print("Max function", np.max(co2))

print("Min method", co2.min())
print("Min function", np.min(co2))

print("Mean method", co2.mean())
print("Mean function", np.mean(co2))

print("Std method", co2.std())
print("Std function", np.std(co2))
```

输出内容如下。

```
Max method 366.84
Max function 366.84
Min method 313.18
Min function 313.18
Mean method 337.053525641
Mean function 337.053525641
Std method 14.9502216262
Std function 14.9502216262
```

（4）中位数（median）可以用 NumPy 或者 SciPy 的函数得到。下列代码将计算数据第 50 百分位数。

```
print("Median", np.median(co2))
print("Score at percentile 50",scoreatpercentile(co2, 50))
```

以下是输出内容。

```
Median 335.17
Score at percentile 50 335.17
```

4.2　用 NumPy 进行线性代数运算

线性代数是数学的一个重要分支，比如，我们可以使用线性代数来解决线性回归问题。子程序包 numpy.linalg 提供了许多线性代数例程，我们可以用它来计算矩阵的逆或特征值，求解线性方程或计算行列式等。对于 NumPy 来说，矩阵可以用 ndarray 的一个子类来表示。

4.2.1　用 NumPy 求矩阵的逆

在线性代数中，假设 A 是一个方阵或可逆矩阵，如果存在一个矩阵 A^{-1}，满足矩阵 A^{-1} 与原矩阵 A 相乘后等于单位矩阵 I 这一条件，那么就称矩阵 A^{-1} 是 A 的逆，相应的数学方程如下。

$$A A^{-1} = I$$

子程序包 numpy.linalg 中的 inv()函数就是用来求矩阵的逆的。下面通过一个例子进行说明，具体步骤如下。

（1）利用 mat()函数创建一个示例矩阵。

```
A = np.mat("2 4 6;4 2 6;10 -4 18")
 print("A\n", A)
```

矩阵 A 的内容如下。

```
A[[ 2 4 6] [ 4 2 6] [10 -4 18]]
```

（2）求矩阵的逆。现在可以利用 inv()子例程来计算逆矩阵了。

```
inverse = np.linalg.inv(A)
 print("inverse of A\n", inverse)
```

逆矩阵显示如下。

```
inverse of A[
[-0.41666667 0.66666667 -0.08333333]
```

```
[ 0.08333333 0.16666667 -0.08333333]
[ 0.25 -0.33333333 0.08333333]]
```

> **提示**
> 如果该矩阵是奇异的，或者非方阵，那么我们就会得到 LinAlgError 消息。如果原意，我们也可以通过手动来验证这个计算结果。这就当作是留给读者的一个作业吧。NumPy 库中的 pinv()函数可以用来求伪逆矩阵，它适用于任意矩阵，包括非方阵。

（3）利用乘法进行验算。

下面我们将 inv()函数的计算结果乘以原矩阵，验算结果是否正确。

```
print("Check\n", A * inverse)
```

不出我们所料，果然得到了一个单位矩阵（当然，前提是一些小误差忽略不计）。

```
Check
[[  1.00000000e+00   0.00000000e+00   -5.55111512e-17]
[ -2.22044605e-16  1.00000000e+00   -5.55111512e-17]
[ -8.88178420e-16  8.88178420e-16    1.00000000e+00]]
```

我们将上面的计算结果减去 3×3 的单位矩阵，会得到求逆过程中出现的误差。

```
print("Error\n", A * inverse - np.eye(3))
```

一般来说，这些误差通常忽略不计，但是在某些情况下，细微的误差也可能导致不良后果。

```
[[ -1.11022302e-16 0.00000000e+00 -5.55111512e-17]
 [ -2.22044605e-16 4.44089210e-16 -5.55111512e-17]
 [ -8.88178420e-16 8.88178420e-16 -1.11022302e-16]]
```

这种情况下，我们需要使用精度更高的数据类型，或者更高级的算法。前面我们使用了 numpy.linalg 子程序包的 inv()例程来计算矩阵的逆。下面，我们用矩阵乘法来验证这个逆矩阵是否符合我们的要求。

```
import numpy as np

A = np.mat("2 4 6;4 2 6;10 -4 18")
```

```
print "A\n", A

inverse = np.linalg.inv(A)
print("inverse of A\n", inverse)

print("Check\n", A * inverse)
print("Error\n", A * inverse - np.eye(3))
```

4.2.2 用 NumPy 解线性方程组

矩阵可以通过线性方式把一个向量变换成另一个向量，因此从数值计算的角度看，这种操作对应于一个线性方程组。Numpy.linalg 中的 solve()子例程可以求解类似 Ax = b 这种形式的线性方程组，其中 A 是一个矩阵，b 是一维或者二维数组，而 x 是未知量。下面我们介绍如何使用 dot()函数来计算两个浮点型数组的点积（dot product）。

这里举例说明解线性方程组的过程，具体步骤如下。

（1）首先创建矩阵 A 和数组 b。

用于创建矩阵 A 和数组 b 代码如下。

```
A = np.mat("1 -2 1;0 2 -8;-4 5 9")
 print("A\n", A)
 b = np.array([0, 8, -9])
 print("b\n", b)
```

矩阵 A 和数组（向量）b 的定义如图 4 - 1 所示。

```
A
[[ 1 -2  1]
 [ 0  2 -8]
 [-4  5  9]]
b
[ 0  8 -9]
```

图 4-1

（2）调用 solve()函数。

（3）接下来，用 solve()函数来解这个线性方程组。

```
x = np.linalg.solve(A, b)
 print("Solution", x)
```

线性方程组的解如下。

```
Solution [ 29. 16. 3.]
```

（4）我们利用 dot()函数进行验算。

我们利用 dot()函数验算这个解是否正确。

```
print("Check\n", np.dot(A , x))
```

结果不出所料。

```
Check[[ 0. 8. -9.]]
```

前面，我们通过 NumPy 的 linalg 子程序包中的 solve()函数求出了线性方程组的解，而且利用 dot()函数验算了结果，下面我们把这些代码放到一起。

```
import numpy as np

A = np.mat("1 -2 1;0 2 -8;-4 5 9")
print("A\n", A)

b = np.array([0, 8, -9])
print("b\n", b)

x = np.linalg.solve(A, b)
print("Solution", x)

print("Check\n", np.dot(A , x))
```

4.3 用 NumPy 计算特征值和特征向量

特征值是方程式 Ax=ax 的标量解（scalar solutions），其中 A 是一个二维矩阵，而 x 是一维向量。特征向量实际上就是表示特征值的向量。

> **提示**
> 特征值和特征向量都是基本的数学概念并且常用于一些重要的算法中，如主成分分析（PCA）算法。PCA 可以极大地简化大规模数据集的分析过程。

计算特征值时，我们可以求助于 numpy.linalg 程序包提供的 eigvals()子例程。函数 eig()的返回值是一个元组，其元素为特征值和特征向量。

我们可以用子程序包 numpy.linalg 的 eigvals()和 eig()函数来获得矩阵的特征值和特征向量，同时通过 dot()函数来验算结果。

```
import numpy as np

A = np.mat("3 -2;1 0")
print("A\n", A)

print("Eigenvalues", np.linalg.eigvals(A))

eigenvalues, eigenvectors = np.linalg.eig(A)
print("First tuple of eig", eigenvalues)
print("Second tuple of eig\n", eigenvectors)

for i in range(len(eigenvalues)):
   print("Left", np.dot(A, eigenvectors[:,i]))
   print("Right", eigenvalues[i] * eigenvectors[:,i])
```

下面来计算一个矩阵的特征值。

（1）我们创建矩阵。

下列代码将创建一个矩阵。

```
A = np.mat("3 -2;1 0")
 print("A\n", A)
```

下面的矩阵即刚才创建的矩阵。

```
A[[ 3 -2] [ 1 0]]
```

（2）利用 eig()函数计算特征值。

这时，我们可以使用 eig()子例程。

```
print("Eigenvalues", np.linalg.eigvals(A))
```

该矩阵的特征值如下。

```
Eigenvalues [ 2. 1.]
```

（3）利用 eig()函数取得特征值和特征向量。

利用 eig()函数，可以得到特征值和特征向量。注意，该函数返回的是一个元组，它的

第 1 个元素是特征值，第 2 个元素为相应的 eigenvectors，以面向列的方式排列。

```
eigenvalues, eigenvectors = np.linalg.eig(A)
 print("First tuple of eig", eigenvalues)
 print("Second tuple of eig\n", eigenvectors)
```

特征值 eigenvalues 和特征向量 eigenvectors 的值分别如下。

```
First tuple of eig [ 2. 1.]
Second tuple of eig
[[ 0.89442719 0.70710678]
[ 0.4472136 0.70710678]]
```

（4）验算结果。

通过 dot()函数计算特征值方程式 Ax = ax 两边的值，我们就可以对结果进行验算。

```
for i in range(len(eigenvalues)):
 print("Left", np.dot(A, eigenvectors[:,i]))
 print("Right", eigenvalues[i] * eigenvectors[:,i])
```

输出内容如下。

```
Left [[ 1.78885438] [ 0.89442719]]
Right [[ 1.78885438] [ 0.89442719]]
Left [[ 0.70710678] [ 0.70710678]]
Right [[ 0.70710678] [ 0.70710678]]
```

4.4　NumPy 随机数

随机数常用于蒙特卡罗法、随机积分等方面。然而，真正的随机数很难获得，实际中使用的都是伪随机数。大部分情况下，伪随机数就足以满足我们的需求。当然，某些特殊情况除外，例如进行高精度的模拟实验时。对于 NumPy，与随机数有关的函数都在 random 子程序包中。

> **提示**
> NumPy 核心的随机数发生器是基于梅森旋转算法的。

我们既可以生成连续分布的随机数，也可以生成非连续分布的随机数。分布函数有一个可选的 size 参数，它能通知 NumPy 要创建多少个数字。我们可以用整型或者元组来给这

个参数赋值，这时会得到相应形状的数组，其值由随机数填充。离散分布包括几何分布、超几何分布和二项式分布。连续分布包括正态分布和对数正态分布。

4.4.1 用二项式分布进行博弈

二项式分布模拟的是在进行整数次独立实验中成功的次数，其中每次实验的成功机会是一定的。

假设我们身在 17 世纪，正在对 8 片币玩法下注。当时流行用 9 枚硬币来玩。如果人头朝上的硬币少于 5 枚，那么我们将输掉一个 8 分币；否则，我们就赢一个 8 分币。下面，我们开始模拟这个游戏，假设我们手上有 1000 硬币作为初始备资。我们将使用 random 模块提供的 binomial()函数进行模拟。

> **提示**
> 完整的代码详见本书代码包中的 ch-04.ipynb 文件。

```
import numpy as np
from matplotlib.pyplot import plot, show

cash = np.zeros(10000)
cash[0] = 1000
outcome = np.random.binomial(9, 0.5, size=len(cash))
for i in range(1, len(cash)):

   if outcome[i] < 5:
      cash[i] = cash[i - 1] - 1
   elif outcome[i] < 10:
      cash[i] = cash[i - 1] + 1
   else:
      raise AssertionError("Unexpected outcome " + outcome)

print(outcome.min(), outcome.max())

plot(np.arange(len(cash)), cash)
show()
```

如果要了解 binomial()函数，请看以下步骤。

（1）我们用 binomial()函数。

我们将数组初始化为零，即现金余额为零。调用 binomial()函数，将 size 参数设为 10 000，这就表示我们要掷 10 000 次硬币。

```
cash = np.zeros(10000)
cash[0] = 1000
outcome = np.random.binomial(9, 0.5, size=len(cash))
```

（2）更新现金余额。

我们利用抛郑硬币的结果来更新 cash 数组，显示 outcome 数组中的最大值和最小值，只要保证没有罕见的异常值即可。

```
for i in range(1, len(cash)):
 if outcome[i] < 5:
 cash[i] = cash[i - 1] - 1
 elif outcome[i] < 10:
 cash[i] = cash[i - 1] + 1
 else:
 raise AssertionError("Unexpected outcome " + outcome)
 print(outcome.min(), outcome.max())
```

不出所料，值为 0～9。

```
0 9
```

（3）用 matplotlib 绘出 cash 数组的图像。

```
plot(np.arange(len(cash)), cash)
show()
```

从图 4-2 可以看出，现金余额的曲线类似于随机行走（即不按照固定模式，而是随机游走）。

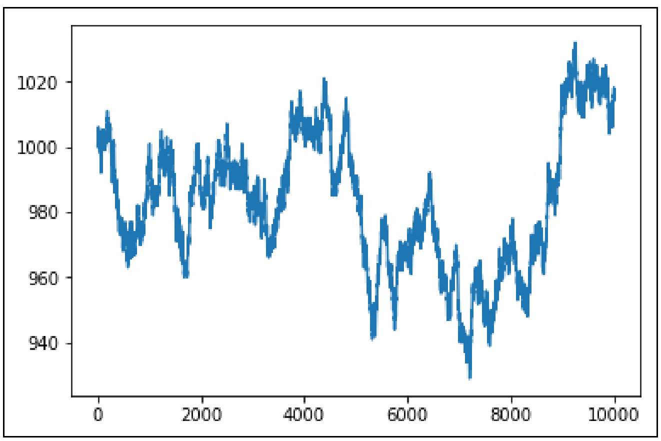


图 4-2

当然，我们每一次执行程序代码，都会得到一个不同的随机游走。如果想总是得到相同的结果，需要给 NumPy 的随机数子程序包中的 binomial()函数一个种子值。

4.4.2 正态分布采样

连续分布是通过概率密度函数（PDF）进行建模的。在特定区间发生某事件的可能性可以通过概率密度函数的积分运算求出。NumPy 的 random 模块提供了许多表示连续分布的函数，如 beta、chisquare、exponential、f、gamma、gumbel、laplace、lognormal、logistic、multivariate_normal、noncentral_chisquare、noncentral_f、normal 等。

利用 NumPy 的 random 子程序包中的 normal()函数，可以把正态分布以直观的形式图示出来。下面给出随机产生的数值的钟形曲线和条形图。

```
import numpy as np
import matplotlib.pyplot as plt

N=10000

normal_values = np.random.normal(size=N)
dummy, bins, dummy = plt.hist(normal_values, np.sqrt(N), normed=True, lw=1)
sigma = 1
mu = 0
plt.plot(bins, 1/(sigma * np.sqrt(2 * np.pi)) * np.exp( - (bins - mu)**2 /
(2 * sigma**2) ),lw=2)
plt.show()
```

随机数可以按照正态分布的要求生成，它们的分布情况可以用条形图来显示。如果要绘制正态分布，请执行以下步骤。

（1）成数值。

借助于 NumPy 的 random 子程序包提供的 normal()函数，可以创建指定数量的随机数。

```
N=100.00
normal_values = np.random.normal(size=N)
```

（2）画出条形图和理论上的 PDF。

下面使用 matplotlib 来绘制条形图和理论上的 PDF，这里中心值为 0，标准差为 1。

```
dummy, bins, dummy = plt.hist(normal_values, np.sqrt(N),
normed=True, lw=1)
sigma = 1
```

```
mu = 0
plt.plot(bins, 1/(sigma * np.sqrt(2 * np.pi)) * np.exp( -
(bins - mu)**2 / (2 * sigma**2) ),lw=2)
plt.show()
```

图 4-3 为我们展示了著名的钟形曲线。

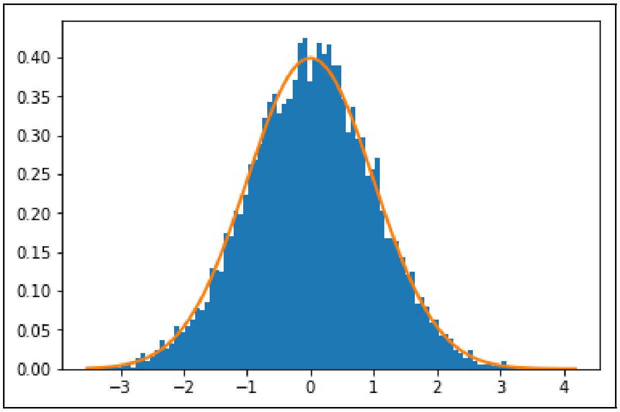


图 4-3

4.4.3　用 SciPy 进行正态检验

正态分布被广泛用于科学和统计学领域，按照中心极限定理，随着独立观测的随机样本数量的增加，它们会趋向呈正态分布。正态分布的特性已经为大家所熟知，并且它还非常便于使用。不过，它需要满足许多必要条件，例如数据点的数量要足够大，同时还要求这些数据点必须是相互独立的。工作中，我们需要养成检查数据是否符合正态分布的好习惯。正态检验的方法有很多，有一些已经在 scipy.stats 程序包中实现了。本节将用到这些检验方法，样本是取自 Google 网站的流感趋势数据。这里我们对原始文件进行了简化处理，只留下日期和来自阿根廷的数据两列，下面给出示例行。

```
Date,Argentina
29/12/02,
05/01/03,
12/01/03,
19/01/03,
26/01/03,
02/02/03,136
```

这些数据可以在本书代码包中的 goog_flutrends.csv 文件中找到，和前面一样，这里我们也按照正态分布对这些数据进行抽样。得到数组的大小与流感趋势数组相同，如果顺

利通过正态检验，就以它为金标准（the golden standard）。这里的代码取自代码包中的 ch-04.ipynb 文件。

```
import numpy as np
from scipy.stats import shapiro
from scipy.stats import anderson
from scipy.stats import normaltest

flutrends = np.loadtxt("goog_flutrends.csv", delimiter=',', usecols=(1,),
skiprows=1, converters = {1: lambda s: float(s or 0)}, unpack=True)
N = len(flutrends)
normal_values = np.random.normal(size=N)

print("Normal Values Shapiro", shapiro(normal_values))
print("Flu Shapiro", shapiro(flutrends))

print("Normal Values Anderson", anderson(normal_values))
print("Flu Anderson", anderson(flutrends))

print("Normal Values normaltest", normaltest(normal_values))
print("Flu normaltest", normaltest(flutrends))
```

作为一个反面例子，下面使用的这个数组跟前面提到的填满零的数组具有同样的大小。实际上，当处理一些小概率事件（如世界性疫情的爆发）时，就可能遇到这种数据。

在这个数据文件中，一些单元是空的。当然，这种问题经常会碰到，所以我们必须习惯于首先清洗数据。我们假定正确的数值是零，下面使用一个转换器来填上这些零值。

```
flutrends = np.loadtxt("goog_flutrends.csv", delimiter=',', usecols=(1,),
skiprows=1, converters = {1: lambda s: float(s or 0)}, unpack=True)
```

夏皮罗-威尔克检验法可以对正态性进行检验。相应的 SciPy 函数将返回一个元组，其中第 1 个元素是一个检验统计量，第 2 个数值是 p 值。需要注意的是，这种填满了零的数组会引发警告。事实上，本例中用到的 3 个函数在处理这个数组方面确实有困难，同时还会引发警告。下面是得到的结果。

```
Normal Values Shapiro (0.9967482686042786, 0.2774980068206787)
Flu Shapiro (0.9351990818977356, 2.2945883254311397e-15)
```

这里得到的 p 值类似于本例中后面的第 3 个检验的结果。分析结果基本上是一样的。

Anderson-Darling 检验可以用来检验正态分布以及其他分布，如指数分布、对数分布和

冈贝尔（Gumbel）分布等。相关的 SciPy 函数涉及一个检验统计量和一个数组，该数组存放了 15、10、5、2.5 和 1 等百分值所对应的临界值。如果该统计量大于显著性水平的临界值，我们就可以断定它不具有正态性。我们将得到下列数值。

```
Normal Values Anderson (0.31201465602225653, array([ 0.572, 0.652, 0.782,
0.912,  1.085]), array([ 15. ,  10. ,   5. ,   2.5,   1. ]))
Flu Anderson (8.258614154768793, array([ 0.572,  0.652,  0.782, 0.912,
1.085]), array([ 15. ,  10. ,   5. ,   2.5,   1. ]))
```

对于这种用零填充的数组，我们没有什么好说的，因为返回的统计量根本就不是一个数值。就像我们期望的那样，这个金标准数组是符合正态分布的。然而，流感趋势数据返回的这个统计量大于所有的有关临界值，因此我们可以肯定它不符合正态分布。这 3 个测试函数中，这个看起来是最容易使用的一个。

SciPy 的 normaltest()函数还实现了 D'Agostino 检验和 Pearson 检验功能。该函数返回的元组和 shapiro()函数一样，也包括一个统计量和 p 值。这里的 p 值是双边卡方概率（two-sided Chi-squared probability），卡方分布是另一种著名的分布，这种检验本身基于偏度和峰度检验的 z 分数。偏度系数用来表示分布的对称程度。因为正态分布是对称的，所以偏度系数为零。峰度系数描述的是分布的形状（尖峰、肥尾）。这种正态分布的峰度系数为 3，超额峰度的系数为 0。以下是本次检验获得的数值。

```
Normal Values normaltest (3.102791866779639, 0.21195189649335339)
Flu normaltest (99.643733363569538, 2.3048264115368721e-22)
```

因为这里处理的是 p 值的概率，所以它越大越好，最好接近 1。对于这种用零填充的数组，会得到很奇怪的结果。不过，因为我们已经得到提示，所以对于这个特殊数组来说，结果是不可信的。此外，如果 p 值大于等于 0.5，我们则认为它具有正态性。对于金标准数组，我们会得到一个更小的数值，这说明我们需要进行更多的观察。这一点作为作业，请读者自行练习。

4.5　创建掩码式 NumPy 数组

数据常常是凌乱的，而且含有空白项或者无法处理的字符，好在掩码式数组可以忽略残缺的或无效的数据点。numpy.ma 子程序包提供的掩码式数组隶属于 ndarray，带有一个掩码。本节我们以 Lena Soderberg 的相片为数据源，而且假设某些数据已被破坏。下面是处理掩码式数组的完整代码，取自本书代码包中的 ch - 04.ipynb 文件。

```
import numpy
import scipy
import matplotlib.pyplot as plt

face = scipy.misc.face()

random_mask = numpy.random.randint(0, 2, size=face.shape)

plt.subplot(221)
plt.title("Original")
plt.imshow(face)
plt.axis('off')

masked_array = numpy.ma.array(face, mask=random_mask)

plt.subplot(222)
plt.title("Masked")
plt.imshow(masked_array)
plt.axis('off')

plt.subplot(223)
plt.title("Log")
plt.imshow(numpy.ma.log(face).astype("float32"))
plt.axis('off')

plt.subplot(224)
plt.title("Log Masked")
plt.imshow(numpy.ma.log(masked_array).astype("float32"))
plt.axis('off')

plt.show()
```

最后，我们来展示原图、原图的对数值、掩码式数组及其对数值。

（1）创建一个掩码。

为了得到一个掩码式数组，必须规定一个掩码。下面将生成一个随机掩码，这个掩码的取值非 0 即 1。

```
random_mask = numpy.random.randint(0, 2, size=face.shape)
```

（2）创建一个掩码式数组。

下面应用该掩码来创建一个掩码式数组。

```
masked_array = numpy.ma.array(face, mask=random_mask)
```

我们得到的图 4-4。

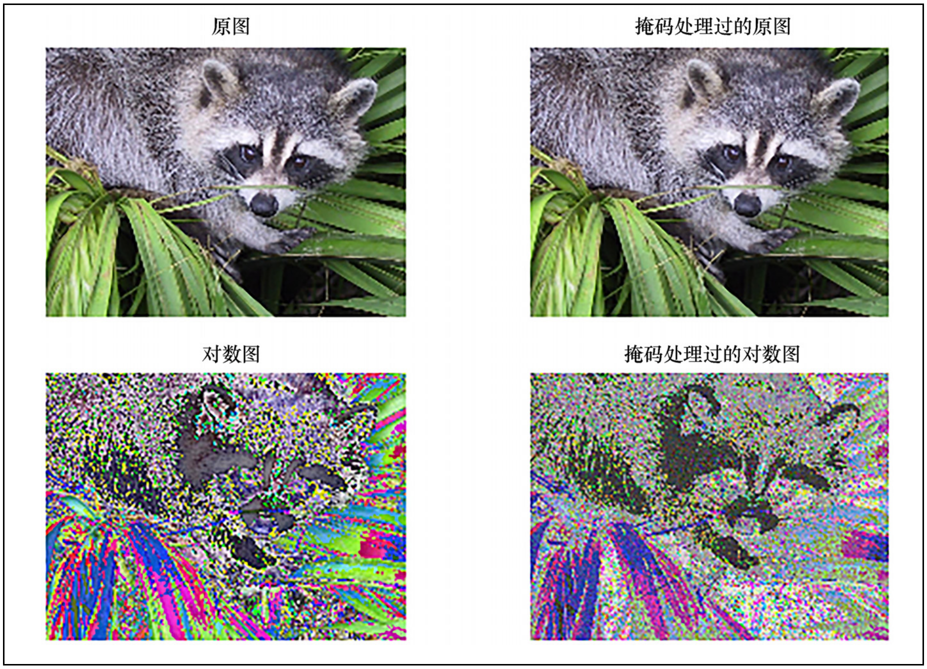


图 4-4

我们给 NumPy 数组附加了一个随机掩码，这样，与该掩码相对应的数据就会被忽略不计。numpy.ma 子程序包为处理掩码式数组提供了各种所需的函数，这里我们只介绍如何生成掩码式数组。

4.6 忽略负值和极值

当希望忽略负值，例如对数组的值取对数时，掩码式数组将会非常有用；此外，剔除异常值时，我们也会用到掩码式数组。这项工作是以极值的上下限为基础的。下面我们以来源于美国职业棒球大联盟（MLB）选手的薪金数据为例，来说明这些技术的应用方法。经过编辑，这份数据仅保留了选手姓名和薪金两栏，结果放在 MLB2008.csv 文件中，读者可以从本书代码包中找到。

至于完整的脚本，请参见本书代码包中的 ch-04.ipynb 文件。

```
import numpy as np
from datetime import date
import sys
```

```
import matplotlib.pyplot as plt

salary = np.loadtxt("MLB2008.csv", delimiter=',', usecols=(1,), skiprows=1,
unpack=True)
triples = np.arange(0, len(salary), 3)
print("Triples", triples[:10], "...")

signs = np.ones(len(salary))
print("Signs", signs[:10], "...")

signs[triples] = -1
print("Signs", signs[:10], "...")

ma_log = np.ma.log(salary * signs)
print("Masked logs", ma_log[:10], "...")

dev = salary.std()
avg = salary.mean()
inside = np.ma.masked_outside(salary, avg - dev, avg + dev)
print("Inside", inside[:10], "...")

plt.subplot(311)
plt.title("Original")
plt.plot(salary)
plt.subplot(312)
plt.title("Log Masked")
plt.plot(np.exp(ma_log))

plt.subplot(313)
plt.title("Not Extreme")
plt.plot(inside)

plt.subplots_adjust(hspace=.9)

plt.show()
```

以下是对上述命令的说明。

（1）对负数取对数

我们可以对含有负数的数组取对数。首先创建一个数组，存放可以被 3 整除的数组。

```
triples = numpy.arange(0, len(salary), 3)
 print("Triples", triples[:10], "...")
```

然后生成一个元素值全为 1 且大小与薪金数据数组相等的数组。

```
signs = numpy.ones(len(salary))
print("Signs", signs[:10], "...")
```

借助于第 2 章中学到的技巧，可以将下标是 3 的倍数的数组元素的值取反。

```
signs[triples] = -1
print("Signs", signs[:10], "...")
```

最终就可以对这个数组取对数了。

```
ma_log = numpy.ma.log(salary * signs)
print("Masked logs", ma_log[:10], "...")
```

下面显示相应的薪金数据（注意，这里有一些数据的值为 NaN）。

```
Triples [ 0 3 6 9 12 15 18 21 24 27] ...
Signs [ 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.] ...
Signs [-1. 1. 1. -1. 1. 1. -1. 1. 1. -1.] ...
Masked logs [-- 14.970818190308929 15.830413578506539 --
13.458835614025542 15.319587954740548 -- 15.648092021712584
13.864300722133706 --] ...
```

（2）忽略极值

此处规定：所谓异常值，就是在平均值一个标准差以下或者在平均值一个标准差以上的那些数值（这个定义未必恰当，只是为了便于计算）。根据上面的定义，可以利用下面的代码来屏蔽极值点。

```
dev = salary.std()
avg = salary.mean()
inside = numpy.ma.masked_outside(salary, avg-dev, avg+dev)
print("Inside", inside[:10], "...")
```

以下代码将显示前 10 个元素。

```
Inside [3750000.0 3175000.0 7500000.0 3000000.0 700000.0
4500000.0 3000000.0 6250000.0 1050000.0 4600000.0] ...
```

下面分别绘制原始薪金数据、取对数后的数据和取幂复原后的数据，最后是应用基于标准差的掩码之后的数据。

具体如图 4-5 所示。

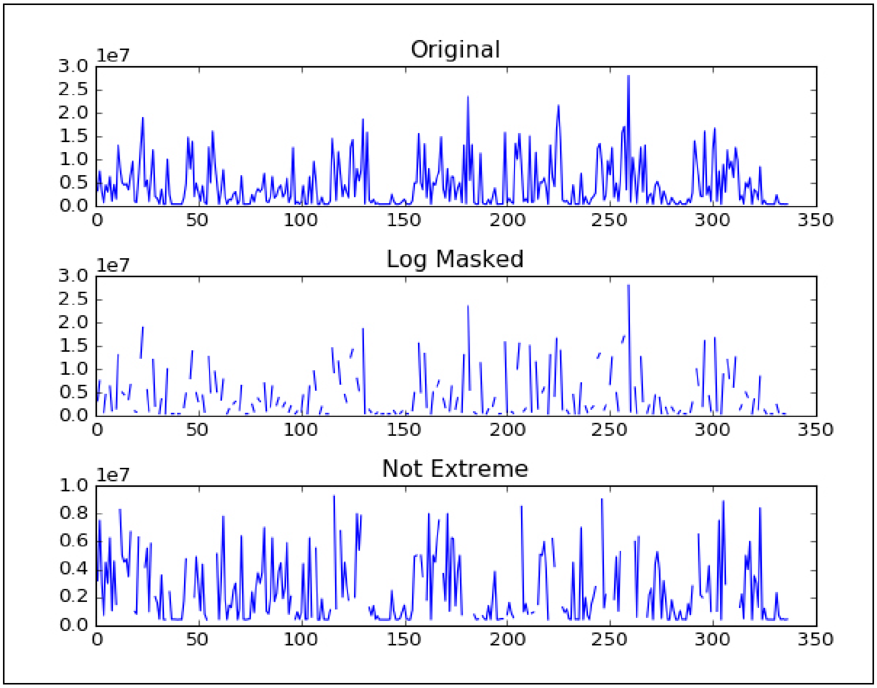


图 4-5

numpy.ma 子程序包中的函数可以屏蔽数组中被视为无效的元素，例如无法应用 log() 和 sqrt()函数的负值元素。被屏蔽的值类似于关系数据库和程序设计中的 NULL 值，对被屏蔽的值进行运算时，给它的都是一个屏蔽后的值。

4.7 小结

本章我们学习了多种 NumPy 和 SciPy 子库，回顾了线性代数、统计学、连续和离散分布、掩码式数组和随机数方面的内容。

第 5 章将介绍更多的技巧，尽管有人可能认为那些工作不算是数据分析。实际上，凡是能够与数据分析扯上关系的东西，我们都需要关注。通常，我们进行数据分析时，并不是在人员配备齐全的团队中工作，所以没人来替我们检索和存储数据。但是，如果要想顺畅地完成数据分析流程，这些工作又是不可或缺的，所以第 5 章会对这些工作进行详尽的说明。

第 5 章
数据的检索、加工与存储

现实中，各种形式的数据随处可见。我们不仅可以从网络、电子邮件和 FTP 中取得数据，也可通过实验研究或者市场调查来获得数据。要想全面总结不同格式数据的获取方法，恐怕要占用大量的篇幅，不是几页就能讲全的。大部分情况下，数据在分析之前或之后我们都需要将其存储起来。关于数据的存储问题，本章也有讨论。第 8 章将讲解各种数据库（关系数据库和 NoSQL 数据库）及其 API 的有关知识。本章探讨的主题如下。

- 利用 NumPy 和 Pandas 对 CSV 文件进行写操作
- 二进制.npy 格式和 pickle 格式
- 利用 PyTables 储存数据
- Pandas DataFrame 与 HDF5 仓库之间的读写操作
- 用 pandas 读写 Excel
- 使用 REST web 服务和 JSON
- 使用 Pandas 读写 JSON
- 解析 RSS 和 Atom 订阅
- 利用 Beautiful Soup 解析 HTML

5.1 利用 NumPy 和 pandas 对 CSV 文件进行写操作

前几章我们学过读取 CSV 文件的内容，其实，对 CSV 文件进行写操作同样也很简单，只不过使用的函数和方法不同罢了。首先，生成一些数据，将来它们会以 CSV 格式保存。

下面的代码给随机数生成器指定种子并生成一个 3×4 的 NumPy 数组。

我们将一个数组元素的值设为 NaN。

```
np.random.seed(42)

a = np.random.randn(3, 4)
a[2][2] = np.nan
print(a)
```

上述代码打印输出的数组如下。

```
[[ 0.49671415 -0.1382643   0.64768854  1.52302986]
 [-0.23415337 -0.23413696  1.57921282  0.76743473]
 [-0.46947439  0.54256004         nan -0.46572975]]
```

NumPy 的 savetxt()函数是与 loadtxt()相对应的一个函数，它能以诸如 CSV 之类的区隔型文件格式保存数组。下面的代码可用来保存刚创建的那个数组。

```
np.savetxt('np.csv', a, fmt='%.2f', delimiter=',', header=" #1, #2, #3, #4")
```

上面的函数调用中，我们规定了用以保存数组的文件的名称、数组、可选格式、间隔符（默认为空格符）和一个可选的标题。

提示

有关格式参数的详细说明，请参考 Python 的官方文档。

通过编辑器，例如在 Windows 系统上的 Notepad 或者 cat 命令，即 cat np.csv，我们可以查看刚才所建的 np.csv 文件的具体内容，具体如下。

```
#  #1,  #2,  #3,  #4
0.50,-0.14,0.65,1.52
-0.23,-0.23,1.58,0.77
-0.47,0.54,nan,-0.47
```

利用随机数组来创建 Pandas DataFrame，代码如下。

```
df = pd.DataFrame(a)
print(df)
```

就像你所看到的那样，Pandas 会自动替我们给数据取好列名。

```
          0         1         2         3
0  0.496714 -0.138264  0.647689  1.523030
```

```
1 -0.234153 -0.234137  1.579213  0.767435
2 -0.469474  0.542560NaN -0.465730
```

利用 Pandas 的 to_csv()方法可以为 CSV 文件生成一个 DataFrame，代码如下。

```
df.to_csv('pd.csv', float_format='%.2f', na_rep="NAN!")
```

对于这个方法，我们需要提供文件名、类似于 NumPy 的 savetxt()函数的格式化参数的可选格式串和一个表示 NaN 的可选字符串。pd.csv 文件的内容如下。

```
,0,1,2,3
0,0.50,-0.14,0.65,1.52
1,-0.23,-0.23,1.58,0.77
2,-0.47,0.54,NAN!,-0.47
```

下面的代码引自本书代码包中的 ch-05.ipynb 文件。

```
import numpy as np
import pandas as pd

np.random.seed(42)

a = np.random.randn(3, 4)
a[2][2] = np.nan
print(a)
np.savetxt('np.csv', a, fmt='%.2f', delimiter=',', header=" #1,  #2,  #3,
#4")
df = pd.DataFrame(a)
print(df)
df.to_csv('pd.csv', float_format='%.2f', na_rep="NAN!")
```

5.2 二进制.npy 与 pickle 格式

大部分情况下，用 CSV 格式来保存文件是一个不错的主意，因为大部分程序设计语言和应用程序都能处理这种格式，交流起来非常方便。然而这种格式有一个缺陷，就是它的存储效率不是很高，原因是 CSV 及其他纯文本格式中含有大量空白符。而后来发明的一些文件格式，如 zip、bzip 和 gzip 等，压缩率则有了显著提升。

以下代码引自本书代码包中的 ch-05.ipynb 文件，它对各种格式的存储利用情况进行了比较。

```
import numpy as np
import pandas as pd
from tempfile import NamedTemporaryFile
from os.path import getsize

np.random.seed(42)
a = np.random.randn(365, 4)

tmpf = NamedTemporaryFile()
np.savetxt(tmpf, a, delimiter=',')
print("Size CSV file", getsize(tmpf.name))

tmpf = NamedTemporaryFile()
np.save(tmpf, a)
tmpf.seek(0)
loaded = np.load(tmpf)
print("Shape", loaded.shape)
print("Size .npy file", getsize(tmpf.name))

df = pd.DataFrame(a)
df.to_pickle(tmpf.name)
print("Size pickled dataframe", getsize(tmpf.name))
print("DF from pickle\n", pd.read_pickle(tmpf.name))
```

NumPy 为自己提供了一种专用的格式，称为.npy，可以用于存储 NumPy 数组。在进一步说明这种格式之前，我们先来生成一个 365×4 的 NumPy 数组并给各个元素填充上随机值。这个数组可以看成是一年中 4 个随机变量的每日观测值的模拟，如一个气象站内传感器测到的温度、湿度、降雨量和气压读数。这里，我们将使用 Python 标准的 NamedTemporaryFile 来存储数据，这些临时文件随后会自动删除。

下面将该数组存入一个 CSV 文件并检查其大小，代码如下。

```
tmpf = NamedTemporaryFile()
np.savetxt(tmpf, a, delimiter=',')
print("Size CSV file", getsize(tmpf.name))
```

这个 CSV 文件的大小如下。

```
Size CSV file 36693
```

首先以 NumPy.npy 格式来保存该数组，随后载入内存并检查数组的形状以及该.npy 文件的大小，具体代码如下。

```
tmpf = NamedTemporaryFile()
np.save(tmpf, a)
tmpf.seek(0)
loaded = np.load(tmpf)
print("Shape", loaded.shape)
print("Size .npy file", getsize(tmpf.name))
```

为了模拟该临时文件的关闭与重新打开过程，我们在上述代码中调用了 seek()函数。数组的形状以及文件大小如下。

```
Shape (365, 4)
Size .npy file 11760
```

不出所料，.npy 文件的大小只有 CSV 文件的 1/3 左右。实际上，利用 Python 可以存储任意复杂的数据结构，也可以用序列化格式来存储 Pandas 的 DataFrame 或者 Series 数据结构。

> **提示**
> 在 Python 中，pickle 是将 Python 对象存储到磁盘或其他介质时采用的一种格式，这个格式化的过程叫作序列化（pickling）。之后我们可以从存储器中重建该 Python 对象，这个逆过程称为反序列化（unpickling），详情请参考 Python 官方文档。序列化技术经过多年的发展，已经出现了多种 pickle 协议。当然，并非所有的 Python 对象都能够序列化；不过，借助诸如 dill 之类的模块，可以将更多种类的 Python 对象序列化。如有可能，最好使用 cPickle 模块（标准 Python 发行版中都含有此模块），因为它是由 C 语言编写的，所以运行起来会更快一些。

首先用前面生成的 NumPy 数组创建一个 DataFrame，接着用 to_pickle()方法将其写入一个 pickle 对象中，然后用 read_pickle()函数从这个 pickle 对象中检索该 DataFrame。

```
df = pd.DataFrame(a)
df.to_pickle(tmpf.name)
print("Size pickled dataframe", getsize(tmpf.name))
print("DF from pickle\n", pd.read_pickle(tmpf.name))
```

该 DataFrame 经过序列化后，尺寸略大于.npy 文件，这一点我们通过下列代码进行确认。

```
Size pickled dataframe 12244
DF from pickle
           0         1         2         3
0   0.496714 -0.138264  0.647689  1.523030
[TRUNCATED]
59 -2.025143  0.186454 -0.661786  0.852433
       ...       ...       ...       ...
[365 rows x 4 columns]
```

5.3 使用 PyTables 存储数据

层次数据格式（Hierarchical Data Format，HDF）是一种存储大型数值数据的技术规范，起源于超级计算社区，目前已经成为一种开放的标准。本书将使用 HDF 的最新版本，也就是 HDF5，该版本仅仅通过组（group）和数据集（dataset）这两种基本结构来组织数据。数据集可以是同类型的多维数组，而组可以用来存放其他组或者数据集。也许读者已经发现了，这里的“组”跟层次式文件系统中的“目录”非常像。

HDF5 最常见的两个主要 Python 程序库如下。

- h5y
- PyTables

本例中使用的是 PyTables。不过，这个程序库需要用到的一些依赖项如下。

- NumPy：第 1 章中已经安装好了 NumPy。
- numexpr：该程序包在计算包含多种运算的数组表达式时，其速度要比 NumPy 快许多倍。
- HDF5。

提示

如果使用 HDF5 的并行版本，则需要安装 MPI。HDF5 可以从网站下载，然后运行下列命令进行安装。这个安装过程可能需要几分钟的时间。具体命令如下。

```
$ gunzip < hdf5-X.Y.Z.tar.gz | tar xf -
$ cd hdf5-X.Y.Z
$ make
$ make install
```

一般情况下，我们使用的程序包管理器都会提供 HDF5，不过我们最好还是选择目前最新的稳定版本（stable version）。截至编写本书期间为止，最新的版本号是 1.8.12。

据 Numexpr 自称，它在某些方面的运算速度要比 NumPy 快得多，因为它支持多线程，而且自己的虚拟机是 C 语言实现的。此外，PyPi 也提供了 Numexpr 和 PyTables，所以我们可以利用 pip 命令来安装它们，具体如下。

```
$ pip3 install numexpr tables
```

此外，我们需要生成一些随机数并用它们来给一个 NumPy 数组赋值。下面我们创建一个 HDF5 文件并把这个 NumPy 数组挂载到根节点上，代码如下。

```
filename = "pytable_demo.h5"
h5file = tables.openFile(filename, mode='w', )
root = h5file.root
h5file.createArray(root, "array", a)
h5file.close()
```

读取这个 HDF5 文件并显示文件大小，代码如下。

```
h5file = tables.openFile(filename, "r")
print(getsize(filename))
```

我们看到，文件大小为 13 824。在读取一个 HDF5 文件并获得该文件的句柄后，就可以通过常规方式来遍历文件，从而找到我们所需的数据。因为这里只有一个数据集，所以遍历起来非常简单。下面的代码将通过 iterNodes()和 read()方法取回 NumPy 数组。

```
for node in h5file.iterNodes(h5file.root):
   b = node.read()
   print(type(b), b.shape)
```

该数据集的形状和类型果然不出我们所料，显示如下。

```
<class 'numpy.ndarray'> (365, 4)
```

以下代码取自本书代码包中的 ch-05.ipynb 文件。

```
import numpy as np
import tables
from os.path import getsize

np.random.seed(42)
```

```
a = np.random.randn(365, 4)

filename = "pytable_demo.h5"
h5file = tables.open_file( filename, mode='w', )
root = h5file.root
h5file.create_array(root, "array", a)
h5file.close()

h5file = tables.open_file(filename, "r")
print(getsize(filename))

for node in h5file.root:
   b = node.read()
   print(type(b), b.shape)

h5file.close()
```

5.4 Pandas DataFrame 与 HDF5 仓库之间的读写操作

HDFStore 类可以看作是 Pandas 中负责 HDF5 数据处理部分的一种抽象。借助一些随机数据和临时文件，我们可以很好地展示这个类的功能特性，具体步骤如下。

我们将临时文件的路径传递给 HDFStore 的构造函数，然后创建一个仓库。

```
filename = "pytable_df_demo.h5"
store = pd.io.pytables.HDFStore(filename)
print(store)
```

上述代码将打印输出该仓库的文件路径及其内容，不过此刻它还没有任何内容。

```
<class 'pandas.io.pytables.HDFStore'>
File path: pytable_df_demo.h5
Empty
```

HDFStore 提供了一个类似字典类型的接口，例如我们可以通过 Pandas 中 DataFrame 的查询键来存储数值。为了将包含随机数据的一个 DataFrame 存储到 HDFStore 中，可以使用下列代码。

```
store['df'] = df
print(store)
```

现在，该仓库存放了如下数据。

```
<class 'pandas.io.pytables.HDFStore'>
File path: pytable_df_demo.h5
            frame          (shape->[365,4])
```

我们可以通过 3 种方式来访问 DataFrame，分别是使用 get()方法访问数据，利用类似字典的查询键访问数据，或者使用点运算符号来访问数据。下面我们分别进行演示。

```
print("Get", store.get('df').shape)
print("Lookup", store['df'].shape)
print("Dotted", store.df.shape)
```

该 DataFrame 的形状同样也可以通过 3 种不同的方式进行访问。

```
Get (365, 4)
Lookup (365, 4)
Dotted (365, 4)
```

为了删除仓库中的数据，我们既可以使用 remove()方法，也可以使用 del 运算符。当然，每个数据项只能删除一次。下面我们从仓库删除 DataFrame，具体代码如下。

```
del store['df']
print("After del\n", store)
```

这个仓库再次变空。

```
After del
<class 'pandas.io.pytables.HDFStore'>
File path: pytable_df_demo.h5
Empty
```

属性 is_open 的作用是指出仓库是否处于打开状态。为了关闭一个仓库，可以调用 close()方法。下面代码展示了关闭仓库的方法并针对仓库的状态进行了相应的检查。

```
print("Before close", store.is_open)
store.close()
print("After close", store.is_open)
```

一旦关闭，该仓库就会退出打开状态，显示如下。

```
Before close True
After close False
```

为读写 HDF 数据，Pandas 还提供了两种方法，一种是 DataFrame 的 to_hdf()方法，另

一种是顶级的 read_hdf()函数。下面示例代码展示了调用 to_hdf()方法读取数据的过程。

```
df.to_hdf(tmpf.name, 'data', format='table')
print(pd.read_hdf(tmpf.name, 'data', where=['index>363']))
```

用于读写操作的应用程序接口的参数包括文件路径、仓库中组的标识符以及可选的格式串。这里的格式有两种，一种是固定格式，另一种是表格格式。固定格式的优点是速度要更快一些，缺点是无法追加数据，也不能进行搜索。表格格式相当于 PyTables 的 Table 结构，可以对数据进行搜索和选择操作。下面是通过查询 DataFrame 得到的数据。

```
            0         1         2         3
364  0.753342  0.381158  1.289753  0.673181
[1 rows x 4 columns]
```

以下代码引自本书代码包中的 ch-05.ipynb 文件。

```
import numpy as np
import pandas as pd

np.random.seed(42)
a = np.random.randn(365, 4)

filename = "pytable_df_demo.h5"
store = pd.io.pytables.HDFStore(filename)
print(store)

df = pd.DataFrame(a)
store['df'] = df
print(store)

print("Get", store.get('df').shape)
print("Lookup", store['df'].shape)
print( "Dotted", store.df.shape)

del store['df']
print("After del\n", store)

print("Before close", store.is_open)
store.close()
print("After close", store.is_open)
df.to_hdf('test.h5', 'data', format='table')
print(pd.read_hdf('test.h5', 'data', where=['index>363']))
```

5.5　使用 Pandas 读写 Excel 文件

现实生活中，许多重要数据都是以 Excel 文件的形式存放的。当然，如果需要，我们也可以将其转换为可移植性更高的诸如 CSV 之类的格式。不过，利用 Python 来操作 Excel 文件会更加方便。在 Python 的世界里，为实现同一目标的项目通常不止一个，如提供 Excel I/O 操作功能的项目就是如此。只要安装了这些模块，我们就能让 Pandas 具备读写 Excel 文件的能力。只是这些方面的说明文档不是很完备，其原因是 Pandas 依赖的这些项目往往各自为战并且发展极为迅猛。这些 Pandas 程序包对于 Excel 文件也很挑剔，要求这些文件的后缀必须是.xls 或者.xlsx；否则就会报错。

```
ValueError: No engine for filetype: ''
```

好在这个问题非常容易解决，举例来说，当创建一个临时文件时，只提供合适的后缀即可。如果需要的多个模块一个都没有安装的话，就会收到如下的错误信息。

```
ImportError: No module named openpyxl.workbook
```

只要用下面的命令安装 openpyxl，就可以杜绝这样的错误提示，具体命令如下。

```
$ pip3 install openpyxl xlsxwriter xlrd
```

模块 openpyxl 源于 PHPExcel，它提供了针对.xlsx 文件的读写功能。

> **小技巧**
> 如果由于某种原因，无法使用 pip install 命令安装模块，我们可以参考 openpyxl 网站上介绍的其他安装方式。

此外，模块 xlsxwriter 也需要读取.xlsx 文件，而模块 xlrd 能用来析取.xls 和.xlsx 文件中的数据。

下面我们先来生成用于填充 Pandas 中 DataFrame 的随机数，然后用这个 DataFrame 创建一个 Excel 文件，接着再用 Excel 文件重建 DataFrame 并通过 mean()方法来计算其平均值。对于 Excel 文件的工作表，我们既可以为其指定一个从 0 开始计数的索引，也可以为其规定一个名称。以下代码引自本书代码包中的 ch-05.ipynb 文件。

```
import numpy as np
```

```
import pandas as pd

np.random.seed(42)
a = np.random.randn(365, 4)

filename="excel_demo.xlsx"
df = pd.DataFrame(a)
print(filename)
df.to_excel(filename, sheet_name='Random Data')
print("Means\n", pd.read_excel(filename, 'Random Data').mean()))
```

我们通过 to_excel()方法创建 Excel 文件，具体如下。

```
df.to_excel(tmpf.name, sheet_name='Random Data')
```

下面我们使用顶级 read_excel()函数来重建 DataFrame，代码如下。

```
print("Means\n", pd.read_excel(tmpf.name, 'Random Data').mean())
```

下面我们输出平均值。

```
/var/folders/k_/xx_xz6xj0hx627654s3vld440000gn/T/tmpeBEfnO.xlsx
Means
0    0.037860
1    0.024483
2    0.059836
3    0.058417
dtype: float64
```

5.6 使用 REST Web 服务和 JSON

表述性状态转移（Representational State Transfer，REST）Web 服务采用的是 REST 架构风格。对于 HTTP（S）来说，可以使用 GET、POST、PUT 和 DELETE 方法，这些方法对应数据项的创建、请求、更新及删除操作。

使用 REST 风格的 API 时，数据项是通过统一资源标识符（URI）进行标识的。虽然 REST 并非官方标准，但是其应用极为广泛，所以我们必须对它进行深入了解。Web 服务经常使用 JavaScript 对象表示法（JavaScript Object Notation，JSON）来交换数据。使用这种格式时，数据会按照 JavaScript 表示法的要求进行加工。这种表示法类似于 Python 的列表和字典的语法。利用 JSON，通过组合列表和字典，可以定义任意复杂的数据。为了解

释这一点，我们将使用一个相当于字典类型的 JSON 字符串为例进行说明，这个字符串用来提供某 IP 地址的地理位置信息。

{"country":"Netherlands","dma_code":"0","timezone":"Europe\/Amsterdam","area_code":"0","ip":"46.19.37.108","asn":"AS196752","continent_code":"EU","isp":"Tilaa V.O.F.","longitude":5.75,"latitude":52.5,"country_code":"NL","country_code3":"NLD"}

以下是引自 ch-5.ipynb 文件中的代码。

```
import json

json_str =
'{"country":"Netherlands","dma_code":"0","timezone":"Europe\/Amsterdam",
"area_code":"0","ip":"46.19.37.108","asn":"AS196752","continent_code":"EU","is
p":"Tilaa
V.O.F.","longitude":5.75,"latitude":52.5,"country_code":"NL","country_code3
":"NLD"}'

data = json.loads(json_str)
print("Country", data["country"])
data["country"] = "Brazil"
print(json.dumps(data))
```

Python 为我们提供了一个简单易用的标准 JSON API。下面我们用 loads()函数来解析 JSON 字符串。

```
data = json.loads(json_str)
```

下列代码可以用来访问变量 country 的值。

```
print "Country", data["country"]
```

上述代码的输出结果如下。

```
Country Netherlands
```

我们修改变量 country 的取值并利用该新 JSON 数据来创建一个字符串。

```
data["country"] = "Brazil"
printjson.dumps(data)
```

得到的这个 JSON 的 country 变量具有一个新值。与字典类似，这里数据项之间的顺序

是任意的。

```
{"longitude": 5.75, "ip": "46.19.37.108", "isp": "Tilaa V.O.F.",
"area_code": "0", "dma_code": "0", "country_code3": "NLD",
"continent_code": "EU", "country_code": "NL", "country": "Brazil",
"latitude": 52.5, "timezone": "Europe/Amsterdam", "asn": "AS196752"}
```

5.7 使用 Pandas 读写 JSON

利用上面例子中的 JSON 字符串，可以轻而易举地创建一个 pandas Series。这个 pandas 提供的 read_json()函数，可以用来创建 pandas Series 或者 pandas DataFrame 数据结构。

以下示例代码引自本书代码包中的 ch-05.ipynb 文件。

```
import pandas as pd

json_str =
'{"country":"Netherlands","dma_code":"0","timezone":"Europe\/Amsterdam",
"area_code":"0","ip":"46.19.37.108","asn":"AS196752","continent_code":"EU","is
p":"Tilaa
V.O.F.","longitude":5.75,"latitude":52.5,"country_code":"NL","country_code3
":"NLD"}'

data = pd.read_json(json_str, typ='series')
print("Series\n", data)

data["country"] = "Brazil"
print("New Series\n", data.to_json())
```

调用 read_json()函数时，既可以向其传递一个 JSON 字符串，也可以为其指定一个 JSON 文件的路径。上面的例子中，我们是利用 JSON 字符串来创建 pandas Series 的。

```
data = pd.read_json(json_str, typ='series')
print("Series\n", data)
```

在得到的 Series 中，键是按字母顺序排列的。

```
Series
area_code                         0
asn                        AS196752
continent_code                   EU
```

```
country                   Netherlands
country_code                       NL
country_code3                     NLD
dma_code                            0
ip                       46.19.37.108
ispTilaa V.O.F.
latitude                         52.5
longitude                        5.75
timezone             Europe/Amsterdam
dtype: object
```

我们再次修改 country 的值并用 to_json()方法将其从 pandas Series 转换成 JSON 字符串。

```
data["country"] = "Brazil"
print("New Series\n", data.to_json())
```

在这个新 JSON 字符串中，键的顺序被保留了下来，不过，country 的值却变了。

```
 New Series
{"area_code":"0","asn":"AS196752","continent_code":"EU","country":"Brazil",
"country_code":"NL","country_code3":"NLD","dma_code":"0","ip":"46.19.37.108
","isp":"Tilaa
V.O.F.","latitude":52.5,"longitude":5.75,"timezone":"Europe\/Amsterdam"}
```

5.8 解析 RSS 和 Atom 订阅

简易信息聚合（Really Simple Syndication，RSS）和 Atom 订阅常用于订阅博客和新闻。虽然这两种订阅的类型不同，却都遵循发布订阅模式。比如，Packt 出版公司网站就提供了书籍和文章方面的订阅功能，只要用户订阅后，就能与网站内容的最新更新保持同步。在 Python 的 feedparser 模块的帮助下，我们无需了解相关的技术细节，就能处理 RSS 和 Atom 订阅。要想安装这个 feedparser 模块，可以使用下列 pip 命令。

```
$ pip3 install feedparser
```

解析完 RSS 文件后，就可以通过句点来访问基础数据了。以下代码将解析 Packt Publishing 网站的订阅源并打印内容数量。

```
import feedparser as fp
```

```
rss = fp.parse("http://www.packtpub.com/rss.xml")

print("# Entries", len(rss.entries))
```

内容的数量如下，不过，每次运行程序时该数字都可能发生变化。

```
# Entries 10
```

这里可以显示含有 Python 这个单词的条目的标题和摘要，代码如下。

```
for i, entry in enumerate(rss.entries):
   if "Python" in entry.summary:
      print(i, entry.title)
      print(entry.summary)
```

就本次运行而言，结果如下。需要注意的是，运行该代码时，内容会有所不同，如果过滤条件限制太严格，还可能一条符合要求的内容也没有。

```
   42 Create interactive plots with matplotlib using Pack't new book and
eBook
    About the author: Alexandre Devert is a scientist. He is an
enthusiastic Python coder as well and never gets enough of it! He used to
teach data mining, software engineering, and research in numerical
optimization.
   Matplotlib is part of the Scientific Python modules collection. It
provides a large library of customizable plots and a comprehensive set of
backends. It tries to make easy things easy and make hard things possible.
It can help users generate plots, add dimensions to plots, and also make
plots interactive with just a few lines of code. Also, matplotlib
integrates well with all common GUI modules.
```

以下代码引自本书代码包中的 ch-05.ipynb 文件。

```
import feedparser as fp

rss = fp.parse("http://www.packtpub.com/rss.xml")

print("# Entries", len(rss.entries))

for i, entry in enumerate(rss.entries):
   if "Java" in entry.summary:
      print(i, entry.title)
      print(entry.summary)
```

5.9 使用 Beautiful Soup 解析 HTML

超文本标记语言（Hypertext Markup Language，HTML）是用于创建网页文档的一种基础性技术。HTML 由各种元素组成，而这些元素由尖括号中的所谓标签组成，如<html>。标签通常是成对出现的，标签对中的第 1 个标签是开始标签，第 2 个标签是结束标签，这些标签以树状结构组织在一起。HTML 的相关规范草案于 1991 年由 Berners-Lee 公布，当时仅包括 18 个 HTML 元素。HTML 的正式定义出现在 1993 年，是由国际互联网工程任务组（Internet Engineering Task Force，IETF）颁布的。1995 年，IETF 发布 HTML 2.0 标准；2013 年，又发行了最新的 HTML 版本，即 HTML5。与 XHTML 和 XML 比较，HTML 不是一个非常严格的标准。

我们知道，现代浏览器容错性已经有了长足进步，另一方面，这也为 Web 页面中不符合标准的非结构化数据的滋生提供了温床。我们不仅可以将 HTML 视为一个硕大的字符串，而且可以运用正则表达式对其执行各种字符串操作。不过，这种方法仅适用于比较简单的项目。

工作中，笔者曾经接触过专业级别的网络信息搜集项目，经验证明我们需要更加先进的方法。在现实世界中，我们有时需要以编程的方式来提交 HTML 表单，尤其是在登录、切换页面和处理 Cookie 时。当我们从网页上抓取数据时，常见的问题是我们无法完全控制所抓取的页面，因此经常需要修改自己的代码。此外，有些网站的所有者不喜欢别人以编程的方式访问其内容，所以他们会处心积虑地设置各种障碍，有的甚至直接禁用这种访问方式。考虑到这些因素，我们应该优先考虑诸如 REST API 之类的信息搜集方法。

如果结果只能通过抓取页面的方式来搜集信息，建议使用 Python 的 Beautiful Soup API。这个应用程序接口不仅可以从 HTML 文件中抽取数据，同时还支持 XML 文件。对于新的项目来说，建议使用 Beautiful Soup 4，因为 Beautiful Soup 3 目前已经停止开发了。我们可以使用如下命令来安装 Beautiful Soup 4。easy_install 命令的用法与此类似。

```
$ pip3 install beautifulsoup4 lxml
```

如果这种方法行不通，还可以把自己的代码跟 Beautiful Soup 直接封装到一起。为了演示如何解析 HTML，本书的代码包中提供了一个名为 loremIpsum.html 的页面文件，该文件是网站 http://loripsum.net/的生成程序制作的。此后，我们对这个文件进行了一些修改。文

件的内容取自公元前 1 世纪西塞罗的拉丁文作品，这是创建网站模型的一种惯用形式。图 5-1 中显示的是网页的上面部分。

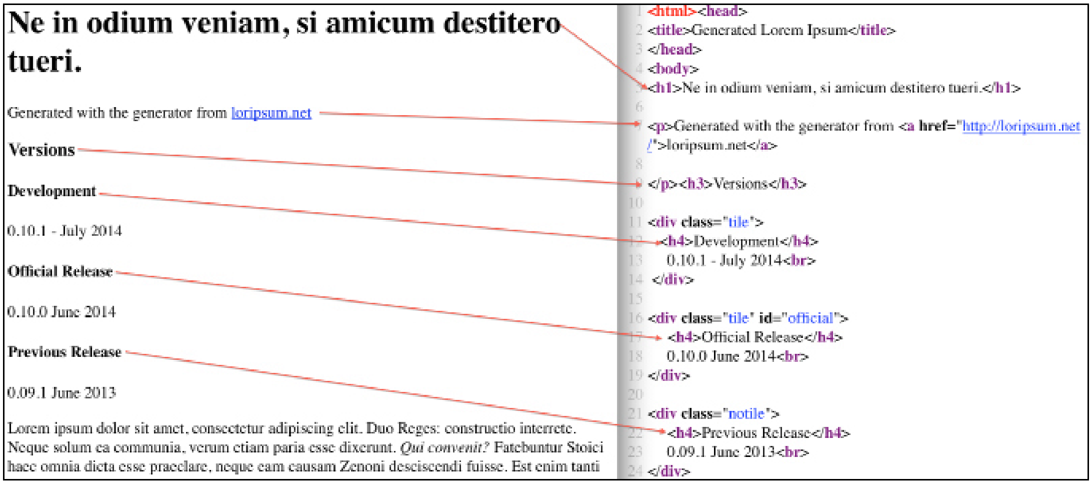


图 5-1

本例中用的工具是 Beautiful Soup 4 和 Python 常规的正则表达式程序库。

以下代码的作用是导入程序库。

```
from bs4 import BeautifulSoup
import re
```

然后我们打开 HTML 文件并新建一个 BeautifulSoup 对象，具体代码如下。

```
soup = BeautifulSoup(open('loremIpsum.html'))
```

通过使用句点标记法，我们可以方便地访问第一个<div>元素，这个元素的作用是组织元素并提供样式。访问第 1 个 div 元素的代码如下。

```
print("First div\n", soup.div)
```

输出内容是一个 HTML 片段，其中可以看到第 1 个<div>标签及其所含内容。

```
First div
<div class="tile">
<h4>Development</h4>
     0.10.1 - July 2014<br/>
</div>
```

> **提示**
>
> 对于这个 div 元素来说，其类属性的值为 tile，这个属性的作用是为该元素指定 CSS 样式。层叠样式表（Cascading Style Sheets，CSS）是一种描述网页元素样式的语言。通过 CSS 类可以方便地控制 Web 页面的外观，因此 CSS 规范的应用极为广泛。有了 CSS，我们可以方便地定义元素的布局、字体和颜色，这对于内容和外观的隔离帮助很大。而内容和外观的隔离，又让设计工作变得更加简单、清晰。

我们可以像访问字典那样来访问标签的属性，下面以输出<div>标签的类属性的值为例进行说明。

```
print("First div class", soup.div['class'])
First div class ['tile']
```

我们利用点号，可以访问任意深度的元素。下面以输出首个<dfn>标签中的文本为例进行说明。

```
print("First dfn text", soup.dl.dt.dfn.text)
```

输出的是一行拉丁文字（英文的意思是：Solisten, I pray）。

```
First dfn text Quareattende, quaeso
```

有时，我们只对 HTML 页面中的超链接感兴趣，例如我们可能只想知道哪些页面具有外向链接。在 HTML 文档中，链接是用<a>标签定义的，通过这个标签的 href 属性，就能找到外向链接的 URL。BeautifulSoup 类中有一个简单易用的 find_all()方法，后面将会经常用到；通过这个方法，我们可以找到文档中所有的超链接。

```
for link in soup.find_all('a'):
      print("Link text", link.string, "URL", link.get('href'))
```

这里，我们从文档中找到了以下 3 个 URL 相同但文字相异的链接。

```
Link text loripsum.net URL http://loripsum.net/
Link text Potera tautem inpune; URL http://loripsum.net/
Link text Is es profecto tu. URL http://loripsum.net/
```

find_all()方法就介绍到这里，下面演示如何访问所有<div>标签中的内容。

```
for i, div in enumerate(soup('div')):
   print(i, div.contents)
```

属性 contents 中存放的是一个 HTML 元素列表。

```
    0 [u'\n', <h4>Development</h4>, u'\n   0.10.1 - July 2014', <br/>,
u'\n']
    1 [u'\n', <h4>Official Release</h4>, u'\n    0.10.0 June 2014', <br/>,
u'\n']
    2 [u'\n', <h4>Previous Release</h4>, u'\n    0.09.1 June 2013', <br/>,
u'\n']
```

为了便于查找，每个标签的 ID 都是唯一的。下面的代码将选取 ID 为 official 的那个<div>元素并打印输出第 3 个元素。

```
official_div = soup.find_all("div", id="official")
print("Official Version", official_div[0].contents[2].strip())
```

许多页面都是根据访问者的输入或者外部数据即时生成的，网上购物网站上的页面尤其如此。当我们跟动态网站打交道时，必须牢记所有标签的属性值随时都可能发生变化。对于大型网站来说，自动生成的 ID 会产生一个大型的字母数字字符串。因此，这时最好不要使用完全匹配方式进行查找，而要使用正则表达式。下面介绍如何通过模式匹配进行查找。上面的代码片段输出的内容是在某个网站上查找一款软件产品时所返回的版本号和月份。

```
Official Version 0.10.0 June 2014
```

我们知道，class 是 Python 编程语言中的一个关键字。为查询标签的类属性，必须使用 class_作为匹配符。下面展示如何求已经定义了类属性的<div>标签的数量。

```
print("# elements with class",len(soup.find_all(class_=True)))
```

如你期望的那样，我们找到了 3 个标签。

```
# elements with class 3
```

下面计算带有“tile”类的<div>标签的数目。

```
tile_class = soup.find_all("div", class_="tile")
print("# Tile classes", len(tile_class))
```

实际上，文档中存在两个含有 tile 类的<div>标签以及 1 个含有 notile 类的<div>标签，因此有如下写法。

```
# Tile classes 2
```

下面定义一个匹配所有<div>标签的正则表达式。

```
print("# Divs with class containing tile", len(soup.find_all("div",
class_=re.compile("tile"))))
```

这次找到了 3 个符合要求的标签。

```
# Divs with class containing tile 3
```

使用 CSS 时，可以利用模式来匹配文档中的元素，这些模式通常称为 CSS 选择器。我们可以利用 BeautifulSoup 类提供的 CSS 选择器来选择页面元素。通过 select()函数，我们可以匹配带有 notile 类的<div>元素如下。

```
print("Using CSS selector\n", soup.select('div.notile'))
```

下面是输出内容。

```
Using CSS selector
[<div class="notile">
<h4>Previous Release</h4>
     0.09.1 June 2013<br/>
</div>]
```

HTML 有序列表看起来与项目编号列表非常接近，由一个<ol>标签和若干<li>标签组成，其中每个列表项对应一个<li>标签。就像 Python 的列表一样，select()函数返回的内容也可以进行切分。图 5-2 展示了有序列表。

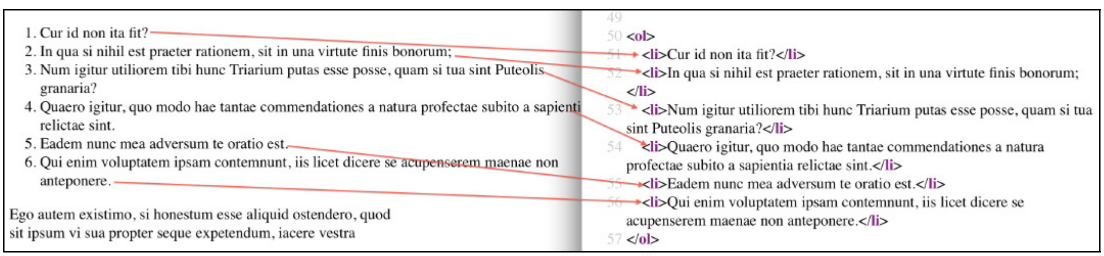


图 5-2

下面的代码将选择有序列表中的前两项内容。

```
print("Selecting ordered list list items\n", soup.select("ol > li")[:2])
```

这两个列表项的内容如下。

```
Selecting ordered list list items
[<li>Cur id non ita fit?</li>, <li>In qua si nihil est praeter
rationem, sit in una virtute finis bonorum;</li>]
```

利用 CSS 选择器的袖珍型语言，可以选择第二个列表项。注意，这里是从 1.开始算起的。具体代码如下。

```
print("Second list item in ordered list", soup.select("ol>li:nth-of-type(2)"))
```

下面是列表中的第二项内容，翻译成英语，大意是“In which, if there is nothing contrary to reason, let him be the power of the end of the good things in one”。

```
Second list item in ordered list [<li>In qua si nihil est praeter
rationem, sit in una virtute finis bonorum;</li>]
```

当用浏览器阅览页面时，可以通过特定的正则表达式来检索匹配的文本节点。以下代码展示了如何借助 text 属性来找出所有包含字符串“2014”的文本节点。

```
print("Searching for text string", soup.find_all(text=re.compile("2014")))
```

得到的文本节点如下。

```
Searching for text string [u'\n   0.10.1 - July 2014', u'\n
0.10.0 June 2014']
```

上面对 BeautifulSoup 类的功能做了简要介绍，此外，Beautiful Soup 还能用于修改 HTML 和 XML 文档。它不仅能够用来排错，还能美化打印效果以及处理不同的字符集。下面的示例代码引自 ch-05.ipynb 文件。

```
from bs4 import BeautifulSoup
import re

soup = BeautifulSoup(open('loremIpsum.html'),"lxml")

print("First div\n", soup.div)
print("First div class", soup.div['class'])

print("First dfn text", soup.dl.dt.dfn.text)
for link in soup.find_all('a'):
   print("Link text", link.string, "URL", link.get('href'))

# Omitting find_all
```

```
for i, div in enumerate(soup('div')):
   print(i, div.contents)

#Div with id=official
official_div = soup.find_all("div", id="official")
print("Official Version", official_div[0].contents[2].strip())

print("# elements with class", len(soup.find_all(class_=True)))

tile_class = soup.find_all("div", class_="tile")
print("# Tile classes", len(tile_class))

print("# Divs with class containing tile", len(soup.find_all("div",
class_=re.compile("tile"))))

print("Using CSS selector\n", soup.select('div.notile'))
print("Selecting ordered list list items\n", soup.select("ol > li")[:2])
print("Second list item in ordered list", soup.select("ol > li:nth-
oftype(2)"))

print("Searching for text string", soup.find_all(text=re.compile("2014")))
```

我们鼓励读者阅读 5.11 节列出的参考资料，以便进一步深入学习 Beautiful Soup 的高级功能，例如搜索返回节点的子节点，获取返回节点的第 n 个父节点，获取返回节点的第 n 个兄弟节点以及其他高级功能。

5.10 小结

本章介绍了检索、加工与存储不同格式数据的方法。这些格式包括 CSV、NumPy .npy、Python pickle、JSON、RSS 和 HTML 等格式。其中，我们用到了 NumPy pandas、JSON、feedparser 以及 Beautiful Soup 等程序库。

第 6 章将为读者讲解利用 Python 显实数据可视化的重要主题。分析数据时，可视化是一种比较常见的任务。它能够展现数据中各个变量之间的关系。通过数据可视化技术，还可以形象地展示出数据的统计特性。

5.11 参考资料

V. G. Nair, *Getting Started with Beautiful Soup*, Packt Publishing, 2014.

第 6 章 数据可视化

数据分析初始阶段，我们通常都要进行可视化处理。借助图示技术，就算枯燥的数值表，也可以展示出它婀娜的一面。数据可视化旨在直观展示信息的分析结果和构思，令某些抽象数据具象化，这些抽象数据包括数据测量单位的性质或数量。数据可视化与科学可视化和统计图示技术关系极为紧密。本章用到的程序库 Matplotlib 是建立在 NumPy 之上的一个 Python 绘图库，它提供了一个面向对象的 API 和一个过程式类 MATLAB API，它们可以并行使用。Matplotlib 使用的图库可以从其官网页面下载。下面我们给出本章涉及的主题。

- Matplotlib 的子库
- Matplotlib 简单绘图
- 对数图
- 散点图
- 图例和注解
- 三维图
- Pandas 绘图
- 时滞图
- 自相关图
- Plot.ly

6.1 Matplotlib 的子库

如果我们对本书代码包中的 ch-01.ipynb 后面部分代码稍作修改，就可以列出 Matplotlib 的各个子库，结果如下。

```
matplotlib version 1.3.1
matplotlib.axes
matplotlib.backends
matplotlib.compat
matplotlib.delaunay DESCRIPTION :Author: Robert Kern
<robert.kern@gmail.com> :Copyright: Copyright 2005
Robert Kern.
:License: BSD-style license. See LICENSE.tx
matplotlib.projections
matplotlib.sphinxext
matplotlib.style
matplotlib.testing
matplotlib.tests
matplotlib.tri
```

这些子程序包的名称基本上不解自明，唯一需要说明的是，这里的后缀 backends 指的是最终结果的输出方式，即可以通过某种格式的文件输出结果，也可以通过图形用户界面输出到屏幕上。

6.2 Matplotlib 绘图入门

第 1 章介绍了 Matplotlib 和 IPython 的安装方法，如果需要，我们可以回头去看。Matplotlib 采用了类似 MATLAB 的过程式 API，这种接口的易用性通常明显优于面向对象的 API，因此，我们首先会演示过程式 API 的使用方法。为了使用 Matplotlib 来绘制基本图形，我们需要调用 matplotlib.pyplot 子库中的 plot()函数。如果数据点具有 x 和 y 坐标，我们就可以使用这个函数来绘制由这种数据点组成的单个或多个数据表的二维图像了。

此外，我们还可以通过格式化参数来指定虚线样式。plot()函数的格式选项和参数非常多，可以通过下面的命令查看（需要提前导入 matplotlib.pyplot 库）。

```
In [1]: help(plot)
```

本例要使用两种样式来绘制线条，即实线（默认样式）和虚线。

下列演示代码引自本书代码包中的 ch-06.ipynb 文件。

```
import matplotlib.pyplot as plt
import numpy as np

x = np.linspace(0, 20)

plt.plot(x, .5 + x)
plt.plot(x, 1 + 2 * x, '--')
plt.show()
```

为了画出上述线条，我们要执行以下步骤。

（1）首先，通过 NumPy 中的 linspace()函数指定横坐标，同时规定起点和终点分别为 0 和 20。

```
x = np.linspace(0, 20)
```

（2）其次通过下列代码画线。

```
plt.plot(x, .5 + x)
plt.plot(x, 1 + 2 * x, '--')
```

（3）此时，既可以用 savefig()函数把图形保存到一个文件中，也可以通过 show()函数将图形显示到屏幕上。将图形显示到屏幕上所需的代码如下。

```
plt.show()
```

最终结果如图 6-1 所示。

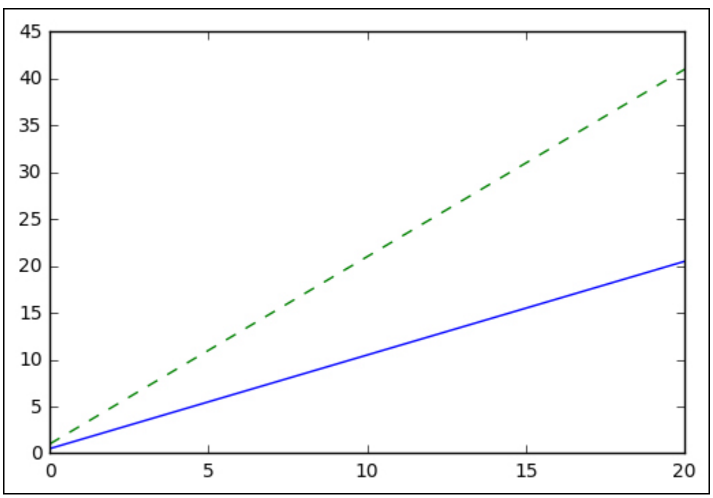


图 6-1

6.3 对数图

所谓对数图，实际上就是使用对数坐标绘制的图形。对于对数刻度来说，其间隔表示的是变量的值在数量级上的变化，这与线性刻度有很大的不同。对数图又分为两种不同的类型，其中一种称为双对数图，它的特点是两个坐标轴都采用对数刻度，对应的 matplotlib 函数是 matplotlib.pyplot.loglog()。另一种，半对数图的一个坐标轴采用线性标度，另一个坐标轴使用对数刻度，它对应的 matplotlib API 是 semilogx()函数和 semilogy()函数。在双对数图上，幂律表现为直线；在半对数图上，直线则代表的是指数律。

摩尔定律就是这样一种增长方式。当然，这不是一条物理定律，而是对经验观测值的一种刻画。摩尔定律是由戈登·摩尔（Gordon Moore）提出来的，大意为集成电路上晶体管的数目，约每两年增加一倍。网上有一个数据表，记录了不同年份微处理器上晶体管的数量。

我们已经为这个数据表制作了一个 CSV 文件，名为 transcount.csv，其中仅含有年份和晶体管数目。此外，我们还需要计算晶体管数量的年度平均值。至于计算平均值和加载该文件的方法，可以由 Pandas 代劳，如果需要，可以回顾第 4 章介绍的方法。求出数据表中晶体管数目的年度平均值后，就可以一条直线来拟合晶体管数量随年份的变化了。NumPy 中的 polyfit()函数可以用多项式来拟合数据。

下列代码引自本书代码包中的 ch-06.ipynb 文件。

```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd

df = pd.read_csv('transcount.csv')
df = df.groupby('year').aggregate(np.mean)
years = df.index.values
counts = df['trans_count'].values
poly = np.polyfit(years, np.log(counts), deg=1)
print("Poly", poly)
plt.semilogy(years, counts, 'o')
plt.semilogy(years, np.exp(np.polyval(poly, years)))
plt.show()
```

下面介绍以上代码。

（1）拟合数据，代码如下。

```
poly = np.polyfit(years, np.log(counts), deg=1)
print("Poly", poly)
```

（2）经过拟合，得到一个 Polynomial 对象，该对象的详情可参考网上的介绍。这个对象的字符串表示，实际上就是多项式系数的按次数降序排列，因此，最高次数项的系数在最前面。就我们的数据而言，得到的多项式系数如下。

```
Poly [ 3.61559210e-01 -7.05783195e+02]
```

（3）NumPy 的 polyval()函数可以用来对上面得到的多项式进行评估。下面我们为数据绘制图像并通过 semilogy()函数进行拟合。

```
plt.semilogy(years, counts, 'o')
plt.semilogy(years, np.exp(np.polyval(poly, years)))
```

这里，实线表示的是趋势线，实心圆表示的是数据点。最终结果如图 6-2 所示。

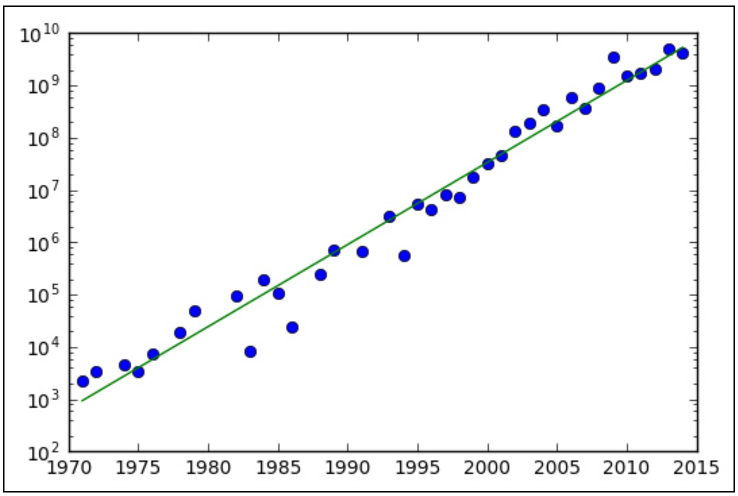


图 6-2

6.4 散点图

散点图可以形象展示直角坐标系中两个变量之间的关系。在散点图中，每个数据点的位置实际上就是两个变量的值。变量之间的任何关系都可以拿散点图来示意。上升趋势模式通常意味着正相关。泡式图是对散点图的一种扩展。在泡式图中，每个数据点都被一个气泡所包围，它由此得名；而第 3 个变量的值正好可以用来确定气泡的

相对大小。

在 WiKi 网站上，也有一个记录图形处理单元（GPU）晶体管数量的数据表。

GPU 是为了高效显示图像而专门设计的。得益于现代显卡的工作机制，GPU 能以高度并行的方式来处理数据。可以说，GPU 是计算技术新时代的弄潮儿。对于本书代码包中的 gpu_transcount.csv 文件，我们没有为它提供太多的数据点。处理缺失数据是泡式图经常要面对的一个问题，对于缺失数据，需要为其定义一个默认的气泡大小。下面，我们再次载入年度数据并计算其平均值，然后通过外部连接操作，按照年份对存放 CPU 和 GPU 晶体管数量的 DataFrame 进行合并。这当中，NaN 值将被设为 0。需要注意的是，就本例而言是可以将 NaN 值设为 0 的，但是其他情况下却未必如此。所有这些功能都已经在第 4 章中讲过了，如有需要，读者可以重新温习一下。为了绘制散点图和泡式图，我们可以借助 matplotlib API 提供的 scatter()函数。要想查阅这个函数的相关说明文档，我们可以使用以下命令。

```
$ ipython3
ln [1]: import matplotlib as mpl
In [2]: help(mpl.scatter)
```

本例中，需要设置参数 s，这个参数与气泡的大小有关。另外，还有一个参数 c，用来指定气泡的颜色。令人遗憾的是，本书非彩色印刷，无法展示气泡的颜色，读者只有亲自运行示例代码，才能看到不同的颜色。这里参数 alpha 的作用是，决定图中气泡的透明度。这个值为 0～1，其中 0 代表完全透明，1 代表完全不透明。创建泡式图的代码如下。

```
plt.scatter(years, cnt_log, c= 200 * years, s=20 + 200 *
gpu_counts/gpu_counts.max(), alpha=0.5)
```

下列代码引自本书代码包中的 ch-06.ipynb 文件。

```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd

df = pd.read_csv('transcount.csv')
df = df.groupby('year').aggregate(np.mean)

gpu = pd.read_csv('gpu_transcount.csv')
gpu = gpu.groupby('year').aggregate(np.mean)

df = pd.merge(df, gpu, how='outer', left_index=True, right_index=True)
df = df.replace(np.nan, 0)
```

```
print(df)
years = df.index.values
counts = df['trans_count'].values
gpu_counts = df['gpu_trans_count'].values
cnt_log = np.log(counts)
plt.scatter(years, cnt_log, c= 200 * years, s=20 + 200 *
gpu_counts/gpu_counts.max(), alpha=0.5)
plt.show()
```

最终结果如图 6-3 所示。

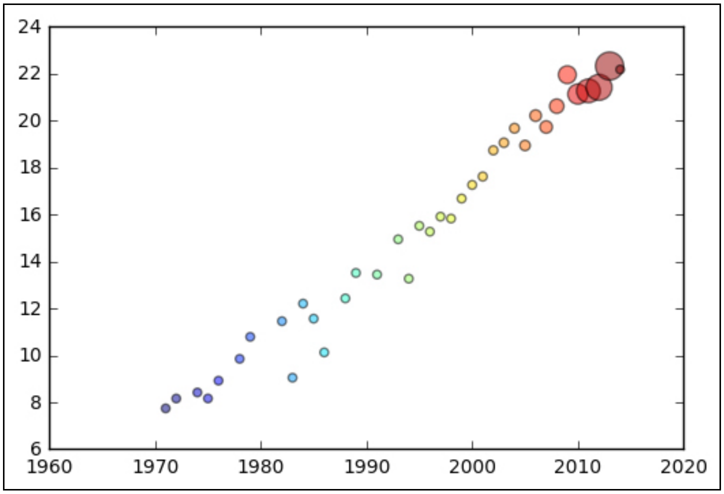


图 6-3

6.5 图例和注解

要想做出让人一眼就懂的“神图”，图例和注解肯定是少不了的。一般情况下，数据图都带有下列辅助信息。

- 用来描述图中各数据序列的图例。为此，可以使用 matplotlib 提供的 legend()函数，来给每个数据序列提供相应的标签。
- 对图中要点的注解。为此，可以借助 matplotlib 提供的 annotate()函数。matplotlib 生成的注解包括标签和箭头两个组成部分。这个函数提供了多个参数，用以描述标签和箭头样式以及其位置。如果想详细了解这些参数，调用 help(annotate)函数即可。
- 横轴和纵轴的标签。这些标签可以通过 xlabel()和 ylabel()函数绘制出来。对于这两

个函数，我们需要提供一个字符串来作为标签的文本，另外还要提供一些可选参数，如标签的字体大小等。

- 一个说明性质的标题，通常由 matplotlib 的 title()函数来提供。一般来说，这个函数只要提供一个描述标题的字符串即可。
- 网格，对于轻松定位数据点非常有帮助。matplotlib 提供的 grid()函数可以用来决定是否启用网格。

下面对上例中的泡式图程序代码稍作修改，加入本章第 2 个例子中的直线。同时，还会为数据序列添加一个标签，相关代码如下。

```
plt.plot(years, np.polyval(poly, years), label='Fit')
plt.scatter(years, cnt_log, c= 200 * years, s=20 + 200 *
gpu_counts/gpu_counts.max(), alpha=0.5, label="Scatter Plot")
```

现在给数据集中的第一个 GPU 添加注释。为此，需要指定相关的数据点，为注解定义标签，指定箭头样式（见参数 arrowprops），同时还要确保把注解置于相应数据点之上。

```
gpu_start = gpu.index.values.min()
y_ann = np.log(df.at[gpu_start, 'trans_count'])
ann_str = "First GPU\n %d" % gpu_start
plt.annotate(ann_str, xy=(gpu_start, y_ann),
arrowprops=dict(arrowstyle="->"), xytext=(-30, +70), textcoords='offset
points')
```

要想阅读完整的代码，请参阅本书代码包中的 ch-06.ipynb 文件。

```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd

df = pd.read_csv('transcount.csv')
df = df.groupby('year').aggregate(np.mean)
gpu = pd.read_csv('gpu_transcount.csv')
gpu = gpu.groupby('year').aggregate(np.mean)

df = pd.merge(df, gpu, how='outer', left_index=True, right_index=True)
df = df.replace(np.nan, 0)
years = df.index.values
counts = df['trans_count'].values
gpu_counts = df['gpu_trans_count'].values

poly = np.polyfit(years, np.log(counts), deg=1)
```

```
plt.plot(years, np.polyval(poly, years), label='Fit')

gpu_start = gpu.index.values.min()
y_ann = np.log(df.at[gpu_start, 'trans_count'])
ann_str = "First GPU\n %d" % gpu_start
plt.annotate(ann_str, xy=(gpu_start, y_ann),
arrowprops=dict(arrowstyle="->"), xytext=(-30, +70), textcoords='offset
points')

cnt_log = np.log(counts)
plt.scatter(years, cnt_log, c= 200 * years, s=20 + 200 *
gpu_counts/gpu_counts.max(), alpha=0.5, label="Scatter Plot")
plt.legend(loc='upper left')
plt.grid()
plt.xlabel("Year")
plt.ylabel("Log Transistor Counts", fontsize=16)
plt.title("Moore's Law & Transistor Counts")
plt.show()
```

最终结果如图 6-4 所示。

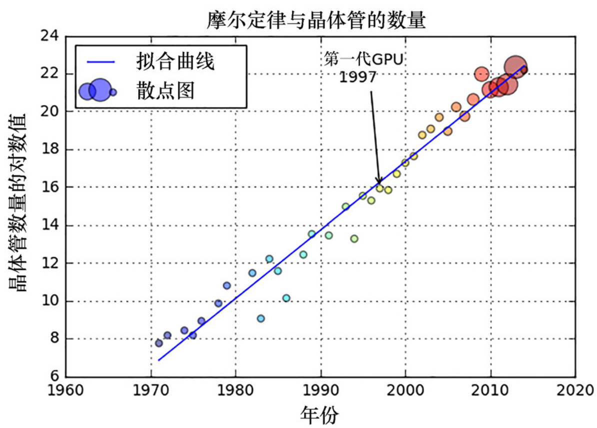


图 6-4

6.6 三维图

对数据可视化来说，二维图像只是家常便饭，要是想来点大餐，那就非三维图形莫属了。笔者负责开发了一个软件包，用它就可以绘制等高线图和三维图。如果我们戴上一副特制眼镜的话，该软件绘图时，图像婉如从我们面前跃出来的一般。

Axes3D 是由 matplotlib API 提供的一个类，可以用来绘制三维图。通过讲解这个类的工作机制，就能够明白面向对象的 matplotlib API 的原理了。matplotlib 的 Figure 类是存放各种图像元素的顶级容器。

（1）首先，创建一个 Figure 对象，代码如下。

```
fig = plt.figure()
```

（2）利用 Figure 对象创建一个 Axes3D 对象。

```
ax = Axes3D(fig)
```

（3）这里，令 x 轴和 y 轴分别表示年份和 CPU 晶体管数量。同时，我们还需要利用存放年份和 CPU 晶体管数量的数组来创建坐标矩阵（coordinate matrices）。创建坐标矩阵时，可以借助 NumPy 中的 meshgrid()函数。

```
X, Y = np.meshgrid(X, Y)
```

（4）通过 Axes3D 类的 plot_surface()方法为数据绘制图像。

```
ax.plot_surface(X, Y, Z)
```

（5）根据面向对象 API 函数的命名约定，应该以 set_开头，以程序对应的函数名结尾，具体如下。

```
ax.set_xlabel('Year')
ax.set_ylabel('Log CPU transistor counts')
ax.set_zlabel('Log GPU transistor counts')
ax.set_title("Moore's Law & Transistor Counts")
```

下列代码摘自本书代码包中的 ch-06.ipynb 文件。

```
from mpl_toolkits.mplot3d.axes3d import Axes3D
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd

df = pd.read_csv('transcount.csv')
df = df.groupby('year').aggregate(np.mean)
gpu = pd.read_csv('gpu_transcount.csv')
gpu = gpu.groupby('year').aggregate(np.mean)

df = pd.merge(df, gpu, how='outer', left_index=True, right_index=True)
```

```
df = df.replace(np.nan, 0)

fig = plt.figure()
ax = Axes3D(fig)
X = df.index.values
Y =
np.where(df['trans_count'].values>0,np.ma.log(df['trans_count'].values), 0)
X, Y = np.meshgrid(X, Y)
Z =
np.where(df['gpu_trans_count'].values>0,np.ma.log(df['gpu_trans_count'].val
ues), 0)
ax.plot_surface(X, Y, Z)
ax.set_xlabel('Year')
ax.set_ylabel('Log CPU transistor counts')
ax.set_zlabel('Log GPU transistor counts')
ax.set_title("Moore's Law & Transistor Counts")
plt.show()
```

最终结果如图 6-5 所示。

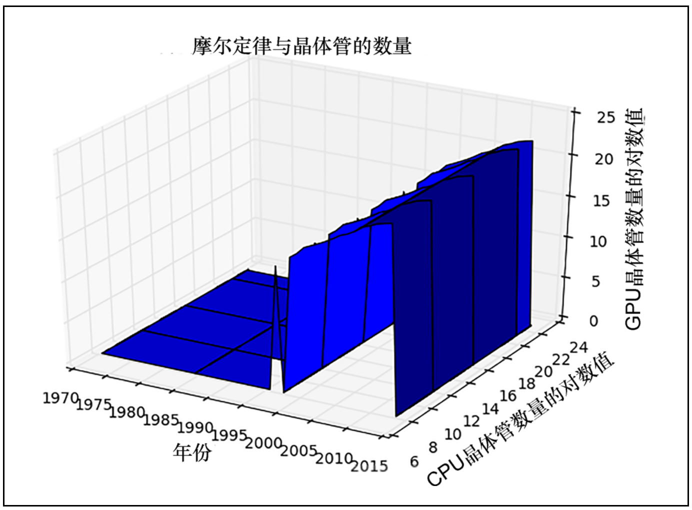


图 6-5

6.7 Pandas 绘图

Pandas 的 Series 类和 DataFrame 类中的 plot()方法都封装了相关的 matplotlib 函数。如果不带任何参数，使用 plot()方法绘制本章一直用的那个数据集，将会得到图 6-6 所示的图像。

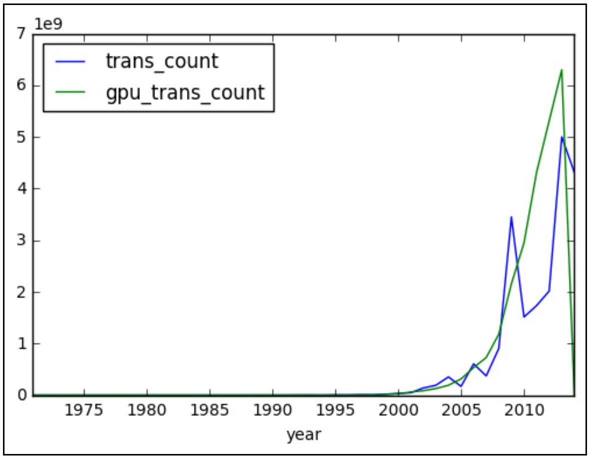


图 6-6

为了创建半对数图，我们需要增设 logy 参数。

```
df.plot(logy=True)
```

我们的数据生成的图像如图 6-7 所示。

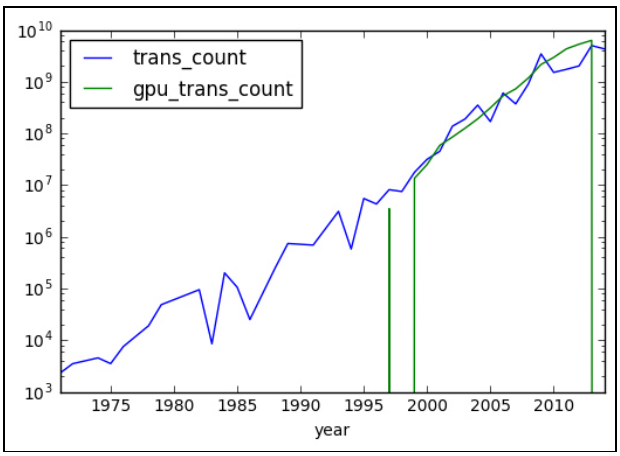


图 6-7

为了创建散点图，我们需要把参数 kind 设为 scatter，同时，还要指定两个列。此外，如果我们将参数 loglog 设为 True，就会生成一个双对数（log-log）图。注意，下列代码要求 Pandas 的版本最低为 0.13.0。

```
df[df['gpu_trans_count'] > 0].plot(kind='scatter', x='trans_count',
y='gpu_trans_count', loglog=True)
```

最终结果如图 6-8 所示。

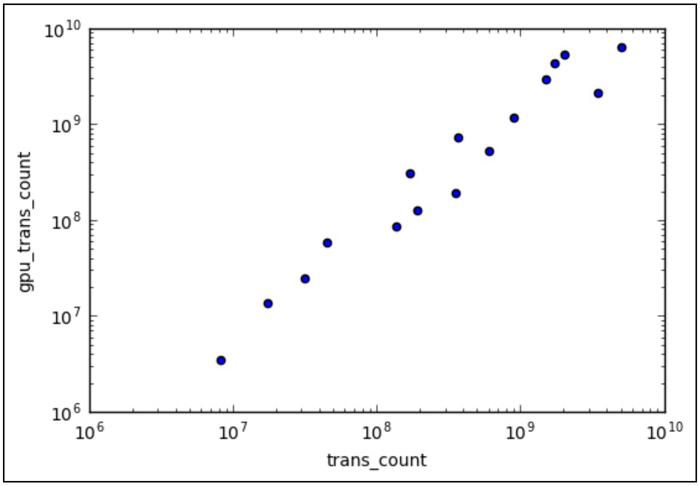


图 6-8

下面的代码摘自本书代码包中的 ch-06.ipynb 文件。

```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd

df = pd.read_csv('transcount.csv')
df = df.groupby('year').aggregate(np.mean)

gpu = pd.read_csv('gpu_transcount.csv')
gpu = gpu.groupby('year').aggregate(np.mean)

df = pd.merge(df, gpu, how='outer', left_index=True, right_index=True)
df = df.replace(np.nan, 0)
df.plot()
df.plot(logy=True)
df[df['gpu_trans_count'] > 0].plot(kind='scatter', x='trans_count',
y='gpu_trans_count', loglog=True)
plt.show()
```

6.8 时滞图

时滞图实际上就是一幅散点图，只不过把时间序列的图像及相同序列在时间轴上后延

的图像放在一起展示而已。举例来说，我们可以利用这种图来考察今年的 CPU 晶体管数量与上一年度 CPU 晶体管数量之间的相关性。我们可以利用 Pandas 子库 pandas.tools.plotting 中的 lag_plot()函数来绘制时滞图。下面是绘制 CPU 晶体管数量时滞图的代码，这里时滞默认为 1，具体如下。

```
lag_plot(np.log(df['trans_count']))
```

最终结果如图 6-9 所示。

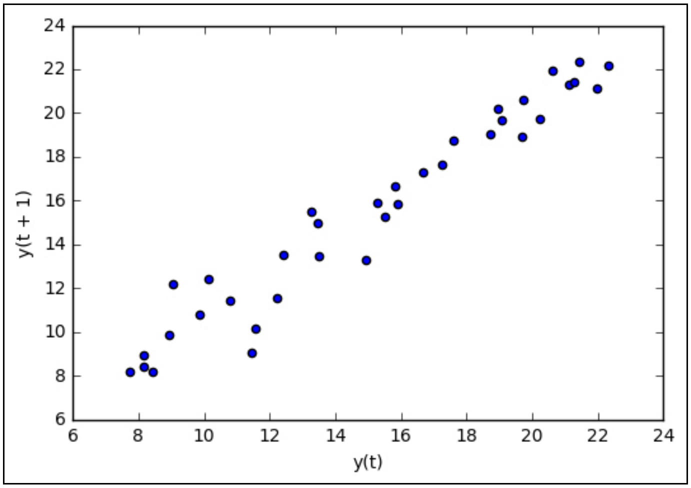


图 6-9

下面是用来演示时滞图的示例代码，摘自本书代码包中的 ch-06.ipynb 文件。

```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from pandas.tools.plotting import lag_plot

df = pd.read_csv('transcount.csv')
df = df.groupby('year').aggregate(np.mean)

gpu = pd.read_csv('gpu_transcount.csv')
gpu = gpu.groupby('year').aggregate(np.mean)

df = pd.merge(df, gpu, how='outer', left_index=True, right_index=True)
df = df.replace(np.nan, 0)
lag_plot(np.log(df['trans_count']))
plt.show()
```

6.9 自相关图

自相关图描述的是时间序列数据在不同时间延迟情况下的自相关性。所谓自相关，用非专业人员的话说，就是时间序列在时刻 n 的值与在时刻 n+l 的值的相互关系，其中 l 就是这里的时间延迟。通常，这些图用于检查时间序列是否具有随机性。就随机时间序列来说，对于所有时间延迟其自相关值接近于零；而对于非随机时间序列，对于某些或所有时间延迟来说自相关值都会具有显著的非零值。我们将在第 7 章中进一步详细说明自相关性。

利用 Pandas 子库 pandas.tools. plotting 中的 autocorrelation_plot()函数，我们就可以画出自相关图了。下列演示代码摘自本书代码包中的 ch-06.ipynb 文件。

```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from pandas.tools.plotting import autocorrelation_plot

df = pd.read_csv('transcount.csv')
df = df.groupby('year').aggregate(np.mean)

gpu = pd.read_csv('gpu_transcount.csv')
gpu = gpu.groupby('year').aggregate(np.mean)

df = pd.merge(df, gpu, how='outer', left_index=True, right_index=True)
df = df.replace(np.nan, 0)
autocorrelation_plot(np.log(df['trans_count']))
plt.show()
```

为 CPU 晶体管数量绘制自相关图的代码如下。

```
autocorrelation_plot(np.log(df['trans_count']))
```

从图 6-10 中可以看出，较之于时间上越远（即时间延迟越大）的数值，当前的数值与时间上越接近（即时间延迟越小）的数值的相关性越大；当时间延迟极大时，相关性衰减为 0。

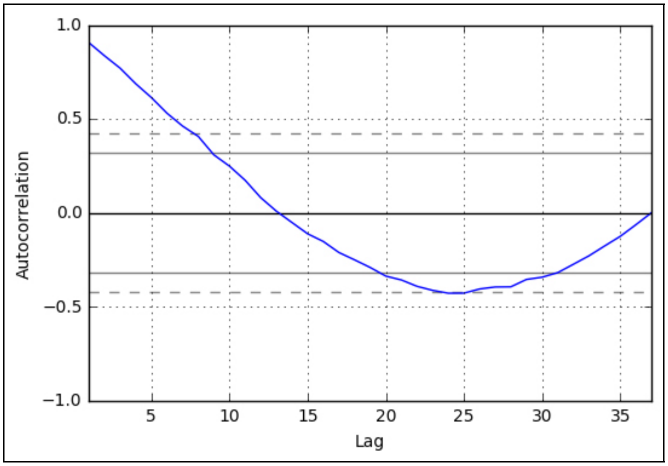


图 6-10

6.10 Plot.ly

Plot.ly 实际上是一个网站，目前处于 beta 测试阶段。它不仅提供了许多数据可视化的在线工具，同时还提供了可在用户机器上使用的对应 Python 库。我们可以通过 Web 接口或以本地方式导入并分析数据，而且还可以将分析结果公布到 Plot.ly 网站上面。团队内的各个成员可以通过这个网站轻松达到共享数据图像的目的，从而实现成员之间的协作，这也正是该网站的独到之处。本节我们将举例说明如何通过 Python API 来绘制箱形图。

箱形图是一种通过四分位数来形象展示数据集的特殊方法。如果把一个有序数据集分为 4 等份，那么第 1 个四分位数就是最小值区间的最大值。第 2 个四分位数是位于数据集中间位置的那个数值，又称“中位数”。第 3 个四分位数是位于中位数和最大值中间位置上的那个数值。箱形图的顶和底分别是第 1 个四分位数和第 3 个四分位数，而贯穿箱子的那条线则是中位数。箱子顶底上的那两根“须”，通常就是数据集的最大值和最小值。本节结束时，我们会提供一幅带有注释的箱形图，届时就会一目了然了。安装 Plot.ly API 的命令如下。

```
$ sudo pip3 install plotly
```

安装 API 后，我们注册获得一个 API 密钥，提供有效密钥后，可以通过下列代码登录。

```
# Change the user and api_key to your own username and api_key
py.sign_in('username', 'api_key')
```

通过 Plot.ly API 创建箱形图的代码如下。

```
data = Data([Box(y=counts), Box(y=gpu_counts)])
plot_url = py.plot(data, filename='moore-law-scatter')
```

下列代码摘自本书代码包中的 ch-06.ipynb 文件。

```
import plotly.plotly as py
from plotly.graph_objs import *
import numpy as np
import pandas as pd

df = pd.read_csv('transcount.csv')
df = df.groupby('year').aggregate(np.mean)

gpu = pd.read_csv('gpu_transcount.csv')
gpu = gpu.groupby('year').aggregate(np.mean)
df = pd.merge(df, gpu, how='outer', left_index=True, right_index=True)
df = df.replace(np.nan, 0)

# Change the user and api_key to your own username and api_key
py.sign_in('username', 'api_key')

counts = np.log(df['trans_count'].values)
gpu_counts = np.log(df['gpu_trans_count'].values)

data = Data([Box(y=counts), Box(y=gpu_counts)])
plot_url = py.plot(data, filename='moore-law-scatter')
print(plot_url)
```

最终结果如图 6-11 所示。

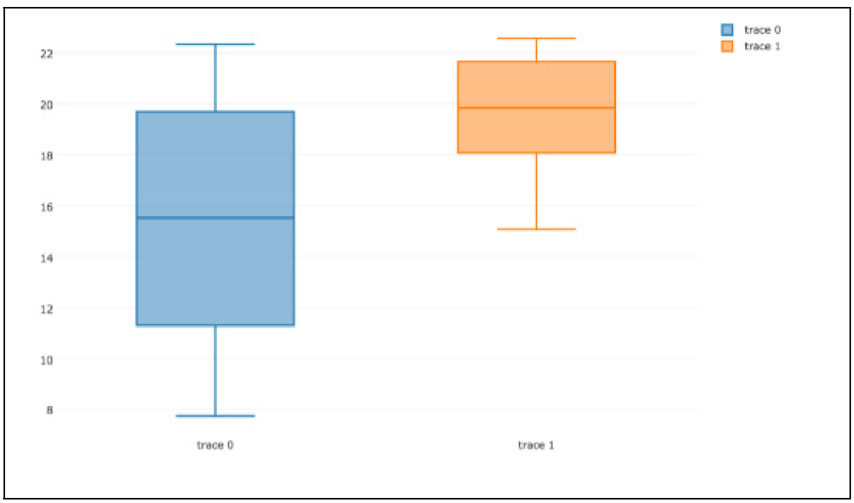


图 6-11

6.11 小结

本章论述了基于 Python 的数据可视化，其中用到了 Matplotlib、Pandas 和 Plot.ly，同时还讲解了箱形图、散点图、泡式图、对数图、自相关图、时滞图、三维图、图例和注解等。

对数图是使用对数坐标绘制的图形；半对数图是一个坐标轴使用线性标度，另一个坐标轴使用对数标度画出的图形。散点图描绘的是两个变量之间的对应关系，而泡式图则是一种特殊的散点图。在泡式图中，第 3 个变量的值表示的是包围数据点的气泡的大小。自相关图描绘的是时间序列数据对于不同时间延迟的自相关性。

同时，我们还介绍了 plot.ly，它是一个用于数据可视化的在线云服务，而且使用该服务创建了一个箱型图。箱型图是根据数据的四分位数来形象展示数据的一种可视化方法。

第 7 章将介绍一种特殊类型的数据：时间序列。时间序列是一些按时间先后顺序排列形成的数据点，当然，这些数据点都提前按照时间戳做了标记。许多物理世界的测量数据都是以时间序列的形式存在的并且一般都被视为信号，如声音信号、光信号或者电信号等。后续章节我们不仅会教大家如何过滤这些信号，而且会介绍如何对时间序列进行建模。

第 7 章
信号处理与时间序列

信号处理是工程和应用数学领域的一个分支，主要用于分析模拟信号和数字信号随时间变化的变量。时间序列分析是信号处理技术的一个分支。时间序列是一个从首次测量开始的数据点的有序列表，数据采样的时间间隔通常是等间距的，如以日或年为间隔进行采样。进行时间序列分析时，数据值的顺序很重要，通常需要设法找出某个值与该序列中之前特定周期数上的另一个数据点或者一组数据点之间的关系。

本章以年度日斑周期数据为例介绍时间序列，这些数据可以从一个开源 Python 项目 statsmodels 程序包中获得。这些例子要用到 NumPy/SciPy、Pandas 和 statsmodels 库。

本章涉及以下主题。

- statsmodels 模块
- 移动平均值
- 窗口函数
- 定义协整
- 自相关
- 自回归模型
- ARMA 模型
- 生成周期信号
- 傅里叶分析
- 谱分析

- 过滤

7.1 statsmodels 模块

如果需安装 statsmodels 模块，请执行以下命令。

```
$ pip3 install statsmodels
```

我们打开代码包内的 ch-07.ipynb 文件，然后修改代码，令其显示 statsmodels 的子库后，能看到如下输出。

```
statmodels version 0.6.1
statsmodels.base
statsmodels.compat
statsmodels.datasets
statsmodels.discrete
statsmodels.distributions
statsmodels.duration
statsmodels.emplike
statsmodels.formula
statsmodels.genmod
statsmodels.graphics
statsmodels.interface
statsmodels.iolib
statsmodels.miscmodels
statsmodels.nonparametric
DESCRIPTION For an overview of this module, see
docs/source/nonparametric.rst PACKAGE CONTENTS _kernel_base
_smoothers_lowess api bandwidths statsmodels.regression
statsmodels.resampling
statsmodels.robust
statsmodels.sandbox
statsmodels.stats
statsmodels.tests
statsmodels.tools
statsmodels.tsa
```

7.2 移动平均值

研究时间序列时，我们经常会用到移动平均值。移动平均法需要规定一个窗口，它限定了每一眼能看到数据的数量，窗口每前移一个周期，其中的数据都要计算一次均值。

$$SMA = \frac{a_m + a_{m-1} + \cdots + a_{m-(n-1)}}{n}$$

不同类型的移动平均法主要区别在于求平均值时所用的权重，例如指数移动平均法，权重随时间的变化以指数的形式递减。

$$EMA_n = EMA_{n-1} + \alpha(p_n - EMA_{n-1})$$

这意味着，数据值的位置越靠前，其对均值的影响力愈弱，有时这种特性正是我们所希望的。

以下代码摘自本书代码包中的 ch-07.ipynb 文件，它将绘制以 11 年和 22 年为窗口的太阳黑子周期的简单移动平均值。

```
import matplotlib.pyplot as plt
import statsmodels.api as sm
from pandas.stats.moments import rolling_mean

data_loader = sm.datasets.sunspots.load_pandas()
df = data_loader.data
year_range = df["YEAR"].values
plt.plot(year_range, df["SUNACTIVITY"].values, label="Original")
plt.plot(year_range, df.rolling(window=11).mean()["SUNACTIVITY"].values,
label="SMA 11")
plt.plot(year_range, df.rolling(window=22).mean()["SUNACTIVITY"].values,
label="SMA 22")
plt.legend()
plt.show()
```

指数移动平均的指数式递减加权策略，可以通过下列 NumPy 代码实现。

```
weights = np.exp(np.linspace(-1., 0., N))
weights /= weights.sum()
```

简单移动平均值使用的是等量加权策略，相应代码如下。

```
def sma(arr, n):
   weights = np.ones(n) / n

   return np.convolve(weights, arr)[n-1:-n+1]
```

因为我们可以把数据载入到 pandas 的 DataFrame 中，所以这样其 rolling_mean()函数使用起来会更加方便。下面我们使用 statsmodels 库加载数据，代码如下。

```
data_loader = sm.datasets.sunspots.load_pandas()
df = data_loader.data
```

最终结果如图 7-1 所示。

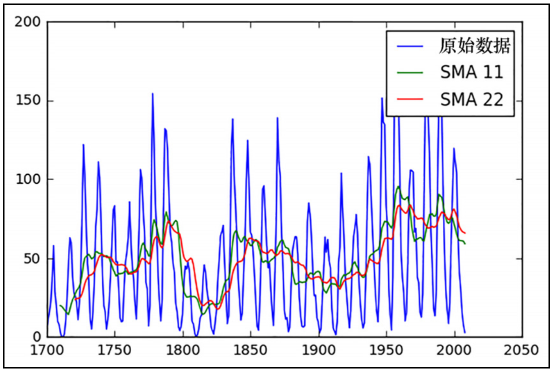


图 7-1

7.3 窗口函数

NumPy 提供了许多窗口例程，当像前面所说的那样滚动窗口时，可以用它们计算权重。

窗口函数是定义在一个区间（窗口）上的函数，超出定义域，函数值取零。我们可以使用它们来分析频谱、设计滤波器等。Boxcar 窗口是一种矩形窗口，公式如下。

```
w(n) = 1
```

三角形窗口的形状像一个三角形，其公式如下。

$$w(n) = 1 - \left| \frac{n - \frac{N-1}{2}}{\frac{L}{2}} \right|$$

上面的公式中，L 可以是 N、$N+1$ 或者 $N-1$。最后介绍的窗口函数是钟形的布莱克曼窗口（Bartlett window），定义如下

$$w(n) = a_0 - a_1 \cos\left(\frac{2\pi n}{N-1}\right) + a_2 \cos\left(\frac{4\pi n}{N-1}\right),$$

其中 $a_0 = \frac{1-a}{2}; a_1 = \frac{1}{2}; a_2 = \frac{a}{2}$

汉宁窗（Hanning window）是另外一种钟形窗口函数，定义如下：

$$w(n) = 0.5\left(1 - \cos\frac{2\pi n}{N-1}\right)$$

根据 Pandas 的应用程序接口的规定，rolling_window()函数的 win_type 参数用来规定窗口函数的类型，另一个参数规定窗口的大小，通常设为 22，这是太阳黑子数据的中等周期（据研究，存在 3 个周期，分别为 11 年、22 年和 100 年）。下面的代码摘自本书代码包中的 ch-07.ipynb 文件，为了便于在图中进行比较，这里只取近 150 年的数据。

```
import matplotlib.pyplot as plt
import statsmodels.api as sm
from pandas.stats.moments import rolling_window
import pandas as pd

data_loader = sm.datasets.sunspots.load_pandas()
df = data_loader.data.tail(150)
df = pd.DataFrame({'SUNACTIVITY':df['SUNACTIVITY'].values},
index=df['YEAR'])
ax = df.plot()

def plot_window(wintype):
    df2 = df.rolling(window=22,win_type=wintype,
center=False,axis=0).mean()
    df2.columns = [wintype]
    df2.plot(ax=ax)

plot_window('boxcar')
plot_window('triang')
plot_window('blackman')
plot_window('hanning')
plot_window('bartlett')
plt.show()
```

最终结果如图 7-2 所示。

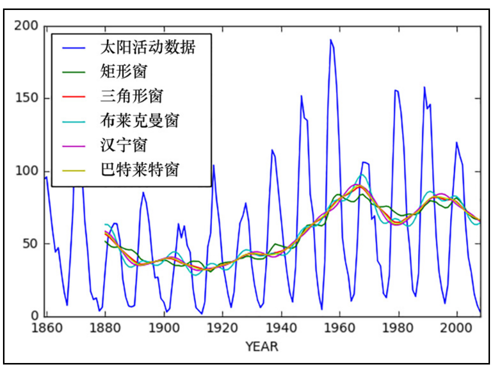


图 7-2

7.4 协整的定义

协整概念类似于关联概念，但是许多人认为，定义两个时间序列相关性时，协整是一种性能优越的衡量指标。如果两个时间序列 x(t)和 y(t)的线性组合是稳态的，那么就称这两个序列具有共整合性或协整性。在这种情况下，下面的方程式应该是稳态的。

```
y(t) - a x(t)
```

考虑醉汉与狗在一起散步的情形，相关性反映出他们是否在同一个方向上前进。协整性反映的则是一段时间后人和狗之间的距离。下面利用随机生成的时间序列和真实数据来展示协整关系。增广迪基-福勒检验法（Augmented Dickey-Fuller test，ADF）可以测试时间序列中的单位根，也可用于确定时间序列的协整关系。

下面的代码摘自本书代码包中的 ch-07.ipynb 文件。

```
import statsmodels.api as sm
from pandas.stats.moments import rolling_window
import pandas as pd
import statsmodels.tsa.stattools as ts
import numpy as np

def calc_adf(x, y):
    result = sm.OLS(x, y).fit()
    return ts.adfuller(result.resid)

data_loader = sm.datasets.sunspots.load_pandas()
data = data_loader.data.values
N = len(data)

t = np.linspace(-2 * np.pi, 2 * np.pi, N)
sine = np.sin(np.sin(t))
print("Self ADF", calc_adf(sine, sine))

noise = np.random.normal(0, .01, N)
print("ADF sine with noise", calc_adf(sine, sine + noise))

cosine = 100 * np.cos(t) + 10
print("ADF sine vs cosine with noise", calc_adf(sine, cosine + noise))

print("Sine vs sunspots", calc_adf(sine, data))
```

下面开始展示协整性。

（1）定义用来计算 ADF 统计量的函数。

```
def calc_adf(x, y):
    result = sm.OLS(x, y).fit()
    return ts.adfuller(result.resid)
```

（2）太阳黑子数据载入 NumPy 数组。

```
data_loader = sm.datasets.sunspots.load_pandas()
data = data_loader.data.values
N = len(data)
```

（3）计算正弦值并求出该值与其自身的协整关系。

```
t = np.linspace(-2 * np.pi, 2 * np.pi, N)
sine = np.sin(np.sin(t))
print("Self ADF", calc_adf(sine, sine))
```

（4）以上代码将打印如下内容。

```
Self ADF (-5.0383000037165746e-16, 0.95853208606005591, 0, 308,
{'5%': -2.8709700936076912, '1%': -3.4517611601803702, '10%':
-2.5717944160060719}, -21533.113655477719)
```

输出的第 1 个值是对 ADF 的度量，第 2 个值是 p 值，这里的 p 值是很高的。接下来是时间延迟和样本量，最后是一个词典，给出了这个样本量的 t 分布值。

（5）下面给正弦波信号添加噪音，看它们是如何影响该信号的。

```
noise = np.random.normal(0, .01, N)
print("ADF sine with noise", calc_adf(sine, sine + noise))
```

（6）混入噪音后，会得到如下所示的结果。

```
ADF sine with noise (-7.4535502402193075, 5.5885761455106898e-
11, 3, 305, {'5%': -2.8710633193086648, '1%':
-3.4519735736206991, '10%': -2.5718441306100512},
-1855.0243977703672)
```

p 值出现明显下降。ADF 指标的值为−7.45，低于字典中所有的临界值，所有这些都是拒绝协整的有力证据。

（7）下面生成一个幅值和偏移量更大的余弦波并混入噪音。

```
cosine = 100 * np.cos(t) + 10
print("ADF sine vs cosine with noise", calc_adf(sine, cosine +
noise))
```

下面是我们得到的值。

```
ADF sine vs cosine with noise (-17.927224617871534, 2.8918612252729532e-30,
16, 292, {'5%': -2.8714895534256861, '1%': -3.4529449243622383, '10%':
-2.5720714378870331}, -11017.837238220782)
```

同样，证据有力地表明拒绝协整。正弦和太阳黑子之间的协整检验结果如下。

```
Sine vs sunspots (-6.7242691810701016, 3.4210811915549028e-09, 16, 292,
{'5%': -2.8714895534256861, '1%': -3.4529449243622383, '10%':
-2.5720714378870331}, -1102.5867415291168)
```

这里所用的两个时间序列对的置信水平大体相当，都与数据点的数量有关，但是变化不大，总结见表 7-1。

表 7-1

序列对	统计量	p 值	5%	1%	10%	拒绝
正弦与正弦	−5.03E-16	0.95	−2.87	−3.45	−2.57	No
正弦与含噪声的正弦	−7.45	5.58E-11	−2.87	−3.45	−2.57	Yes
正弦与含噪声的余弦	−17.92	2.89E-30	−2.87	−3.45	−2.57	Yes
正弦与太阳黑子	−6.72	3.42E-09	−2.87	−3.45	−2.57	Yes

7.5 自相关

自相关是数据集内部的相关性，可用来指明趋势。

对于给定的时间序列，我们只要知道其均值和标准差，就可以用期望值算子来定义时间 s 和 t 的自相关。

$$\frac{E\left[(x_t-\mu_t)(x_s-\mu_s)\right]}{\sigma_t\sigma_s}$$

本质上，这就是把相关性公式应用于一个时间序列及其同一个时间序列的滞后部分。

举例来说，如果后延一个周期，就可以检测前一个值是否影响当前值。当然，如果是

真的，那么计算出的自相关的取值自然会相当高。

第 6 章曾经用 Pandas 函数绘制过自相关图形。本例中将使用 NumPy 库的 correlate() 函数来计算太阳黑子周期实际的自相关值。最后，对取得的数值进行正则化处理。

NumPy 库的 correlate()函数用法如下。

```
y = data - np.mean(data)
norm = np.sum(y ** 2)
correlated = np.correlate(y, y, mode='full')/norm
```

此外，我们对关联度最高值的索引也很感兴趣，这些索引可用 NumPy 的 argsort()函数取得，它返回的是数组排序后对应的下标。

```
print np.argsort(res)[-5:]
```

下面是自相关程度最高的值的下标。

```
[ 9 11 10 1 0]
```

自相关的最大值对应于零延迟，即信号与其自身的相关性；次最大值对应于一个周期的延迟，即 10 年。以下代码摘自本书代码包的 ch-07.ipynb 文件。

```
import numpy as np
import pandas as pd
import statsmodels.api as sm
import matplotlib.pyplot as plt
from pandas.tools.plotting import autocorrelation_plot

data_loader = sm.datasets.sunspots.load_pandas()
data = data_loader.data["SUNACTIVITY"].values
y = data - np.mean(data)
norm = np.sum(y ** 2)
correlated = np.correlate(y, y, mode='full')/norm
res = correlated[len(correlated)/2:]

print(np.argsort(res)[-5:])
plt.plot(res)
plt.grid(True)
plt.xlabel("Lag")
plt.ylabel("Autocorrelation")
plt.show()
autocorrelation_plot(data)
```

```
plt.show()
```

最终结果如图 7-3 所示。

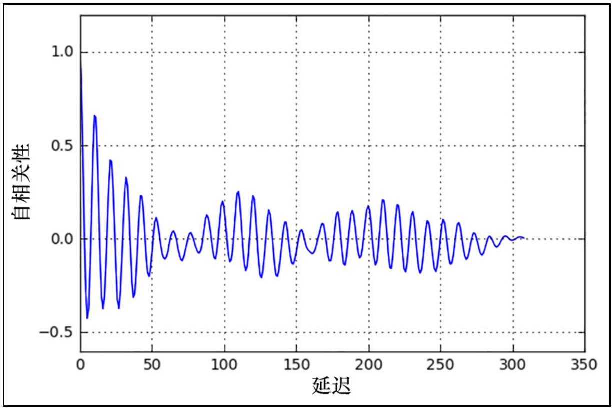


图 7-3

可以将图 7-3 与下面 Pandas 绘制的图 7-4 做比较。

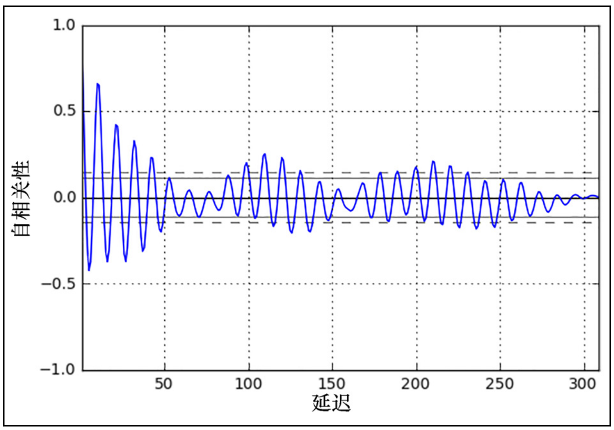


图 7-4

7.6 自回归模型

自回归模型可用于预测时间序列将来的值。使用该模型时，通常需要假定一个随机变量的值依赖于它前面的值。另外，该模型还假定前后值之间的关系是线性的，我们要做的就是拟合数据，以便给数据找到适当的参数。

自回归模型的数学公式如下。

$$x_t = c + \sum_{i=1}^{p} a_i x_{t-i} + \in_t$$

上面公式中，c 是常量，最后一项是随机分量，又名白噪声。

这给我们提出了一个很常见的线性回归问题，但从实用性考虑，保持模型的简单性是十分重要的，因此我们只保留必要的滞后分量。按机器学习的专业术语来说，这些叫作特征。处理回归问题时，Python 的机器学习库 scikit-learn 虽然不是最好的，也是一个上乘之选。第 10 章我们将会用到这个 API。

进行回归分析时，我们常常遇到过拟合的问题，这个问题经常出现在对样本的拟合程度非常理想的情况下，这时一旦引入新的数据点，其表现立刻变差。对付这个问题的标准解决方案是进行交叉验证，或者使用没有过拟合问题的算法。利用这种方法，我们只将一部分样本用于模型参数的估算，其余数据用于该模型的测试和评估。这实际上是一种简化的解释，现实中有更复杂的交叉验证方案，其中很多已在 scikit-learn 中得到支持。为了评估该模型，我们需要计算适当的评价指标。这种评价指标不仅很多，而且随着从业人员对其理解的不断调整，指标的定义也在不断变化。我们可以借助书本或 Wikipedia 找到这些指标的定义，但是请记住，预测或者拟合的评估方法不是一门精确的科学。事实上，指标如此多，只能表明谁也说服不了谁。

下面我们通过 scipy.optimize.leastsq()函数来搭建模型，该函数使用的前两个滞后分量我们在之前已经见过。此外，我们还可以选择一个线性代数函数作为替代。可是，leastsq()函数具有更大的灵活性，让我们几乎可以规定任意类型的模型。搭建模型的代码如下。

```
def model(p, x1, x10):
   p1, p10 = p
   return p1 * x1 + p10 * x10

def error(p, data, x1, x10):
   return data - model(p, x1, x10)
```

拟合模型时，我们需要给参数表赋初值并将其传递给 leastsq()函数，具体如下。

```
def fit(data):
   p0 = [.5, 0.5]
   params = leastsq(error, p0, args=(data[10:], data[9:-1], data[:-10]))[0]
   return params
```

下面在部分数据上训练该模型。

```
cutoff = .9 * len(sunspots)
params = fit(sunspots[:cutoff])
print "Params", params
```

以下是得到的参数。

```
Params [ 0.67172672 0.33626295]
```

有了这些参数，就可以绘出预测值并计算各个指标。下面是得到的指标值。

```
Root mean square error 22.8148122613
Mean absolute error 17.6515446503
Mean absolute percentage error 60.7817800736
Symmetric Mean absolute percentage error 34.9843386176
Coefficient of determination 0.799940292779
```

最终结果如图 7-5 所示。

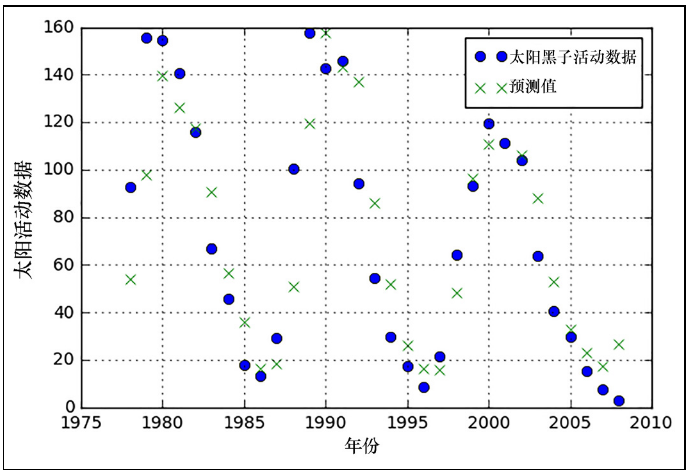


图 7-5

许多预测看起来几乎命中，而另一些则相去甚远。总的来说，拟合效果不是很理想，但也不算很烂，介于两者之间。

以下代码摘自本书代码包中的 ch-07.ipynb 文件。

```
from scipy.optimize import leastsq
import statsmodels.api as sm
import matplotlib.pyplot as plt
import numpy as np
```

```
def model(p, x1, x10):
   p1, p10 = p
   return p1 * x1 + p10 * x10

def error(p, data, x1, x10):
   return data - model(p, x1, x10)

def fit(data):
   p0 = [.5, 0.5]
   params = leastsq(error, p0, args=(data[10:], data[9:-1], data[:-10]))[0]
   return params

data_loader = sm.datasets.sunspots.load_pandas()
sunspots = data_loader.data["SUNACTIVITY"].values
cutoff = .9 * len(sunspots)
params = fit(sunspots[:cutoff])
print("Params", params)

pred = params[0] * sunspots[cutoff-1:-1] + params[1] *
sunspots[cutoff-10:-10]
actual = sunspots[cutoff:]
print("Root mean square error", np.sqrt(np.mean((actual - pred) ** 2)))
print("Mean absolute error", np.mean(np.abs(actual - pred)))
print("Mean absolute percentage error", 100 * np.mean(np.abs(actual -
pred)/actual))
mid = (actual + pred)/2
print("Symmetric Mean absolute percentage error", 100 *
np.mean(np.abs(actual - pred)/mid))
print("Coefficient of determination", 1 - ((actual - pred) ** 2).sum()/
((actual - actual.mean()) ** 2).sum())
year_range = data_loader.data["YEAR"].values[cutoff:]
plt.plot(year_range, actual, 'o', label="Sunspots")
plt.plot(year_range, pred, 'x', label="Prediction")
plt.grid(True)
plt.xlabel("YEAR")
plt.ylabel("SUNACTIVITY")
plt.legend()
plt.show()
```

7.7 ARMA 模型

ARMA 模型由自回归模型和移动平均模型结合而成，常用于时间序列的预测。使用移动平均模型时，我们通常假定随机变量为噪声分量的线性组合与时间序列的均值之和。

提示

自回归模型和移动平均模型可以具有不同的阶数。一般来说，我们能够定义一个具有 p 个自回归项和 q 个移动平均项的 ARMA 模型，如下所示。

$$x_t = c + \sum_{i=1}^{p} a_i x_{t-i} + \sum_{i=1}^{q} b_i \varepsilon_{t-i} \in_t$$

正如自回归模型公式那样，上面的公式中也含有常数部分和白噪声部分。然而，这里还要设法拟合后面的噪声部分。

幸运的是，我们可以使用 statsmodelssm.tsa.ARMA()例程进行此类分析。下面使用 ARMA(10,1)模型来拟合数据，代码如下。

```
model = sm.tsa.ARMA(df, (10,1)).fit()
```

我们进行预测（statsmodels 模块使用了许多字符串）。

```
prediction = model.predict('1975', str(years[-1]), dynamic=True)
```

最终结果如图 7-6 所示。

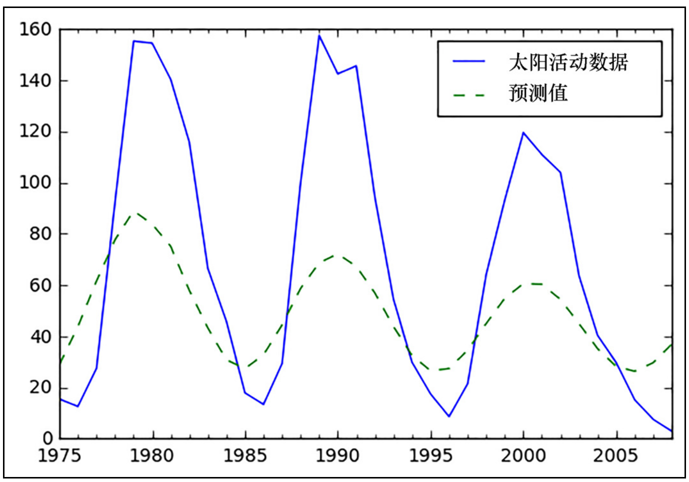


图 7-6

坦白地说，拟合效果很差，原因是该模型对数据过拟合了。在 7.6 节中，这个简单模型的表现要更好一些。下面的示例代码摘自本书代码包中的 ch-07.ipynb 文件。

```
import pandas as pd
import matplotlib.pyplot as plt
import statsmodels.api as sm
import datetime

data_loader = sm.datasets.sunspots.load_pandas()
df = data_loader.data
years = df["YEAR"].values.astype(int)
df.index = pd.Index(sm.tsa.datetools.dates_from_range(str(years[0]),
str(years[-1])))
del df["YEAR"]

model = sm.tsa.ARMA(df, (10,1)).fit()
prediction = model.predict('1975', str(years[-1]), dynamic=True)

df['1975':].plot()
prediction.plot(style='--', label='Prediction')
plt.legend()
plt.show()
```

7.8 生成周期信号

许多自然现象就像精确的时钟一样，具有规律性，并且值得信赖。而有些自然现象，则会表现出一些看上去非常规则的模式。通过希尔伯特-黄变换（Hilbert-Huang transform），一个科学家小组发现太阳黑子活动具有 3 个不同的周期。这 3 个周期的持续时间大致为 11 年、22 年和 100 年。一般情况下，我们使用正弦函数之类的三角函数来模拟周期信号。当然，这些函数我们在中学时就学过了。对于本例来说，这些知识就足够了。因为这里有 3 个周期，所以看上去通过 3 个正弦函数线性组合成一个模型比较合理。这种方法只需要对自回归模型的代码稍作修改。下列代码摘自本书代码包中的 periodic.py 文件。

```
from scipy.optimize import leastsq
import statsmodels.api as sm
import matplotlib.pyplot as plt
import numpy as np
def model(p, t):
   C, p1, f1, phi1 , p2, f2, phi2, p3, f3, phi3 = p
```

```
    return C + p1 * np.sin(f1 * t + phi1) + p2 * np.sin(f2 * t + phi2) +p3 *
np.sin(f3 * t + phi3)


def error(p, y, t):
    return y - model(p, t)

def fit(y, t):
    p0 = [y.mean(), 0, 2 * np.pi/11, 0, 0, 2 * np.pi/22, 0, 0, 2 *
np.pi/100, 0]
    params = leastsq(error, p0, args=(y, t))[0]
    return params
data_loader = sm.datasets.sunspots.load_pandas()
sunspots = data_loader.data["SUNACTIVITY"].values
years = data_loader.data["YEAR"].values

cutoff = .9 * len(sunspots)
params = fit(sunspots[:cutoff], years[:cutoff])
print("Params", params)

pred = model(params, years[cutoff:])
actual = sunspots[cutoff:]
print("Root mean square error", np.sqrt(np.mean((actual - pred) ** 2)))
print("Mean absolute error", np.mean(np.abs(actual - pred)))
print("Mean absolute percentage error", 100 * np.mean(np.abs(actual -
pred)/actual))
mid = (actual + pred)/2
print("Symmetric Mean absolute percentage error", 100 *
np.mean(np.abs(actual - pred)/mid))
print("Coefficient of determination", 1 - ((actual - pred) ** 2).sum()/
((actual - actual.mean()) ** 2).sum())
year_range = data_loader.data["YEAR"].values[cutoff:]
plt.plot(year_range, actual, 'o', label="Sunspots")
plt.plot(year_range, pred, 'x', label="Prediction")
plt.grid(True)
plt.xlabel("YEAR")
plt.ylabel("SUNACTIVITY")
plt.legend()
plt.show()
```

输出内容如下。

```
Params [ 47.18800285 28.89947419 0.56827284 6.51168446 4.55214999
```

```
0.29372077 -14.30926648 -18.16524041 0.06574835 -4.37789602]
Root mean square error 59.5619175499
Mean absolute error 44.5814573306
Mean absolute percentage error 65.1639657495
Symmetric Mean absolute percentage error 78.4477263927
Coefficient of determination -0.363525210982
```

第一行展示的是模型的相关系数。这里的平均绝对误差是 44，表示各个预测值与真实值之间平均相差多大。为了更好地拟合数据，判定系数应尽量接近 1。可是，实际上得到的判定系数却是一个负值，这离我们的要求相去甚远。最终结果如图 7-7 所示。

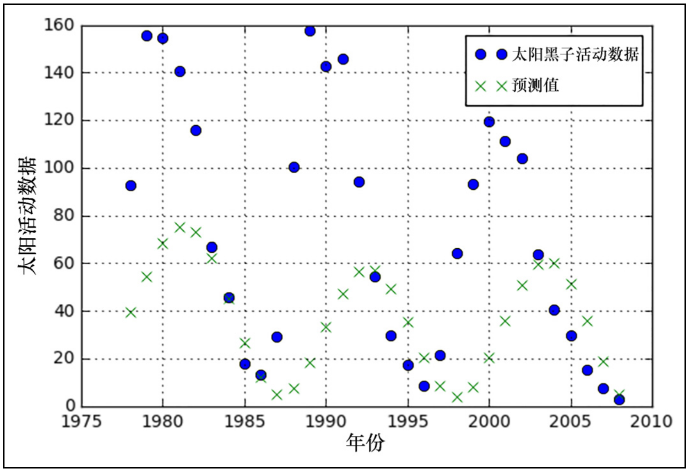


图 7-7

7.9 傅里叶分析

傅里叶分析是建立在以数学家 Joseph Fourier 命名的傅里叶级数之上的一种数学方法。傅里叶级数是一种表示函数的数学方法，它通常使用正弦函数和余弦函数构成的无穷级数来表示函数。这些函数既可以是实值函数，也可以是虚值函数。

$$\sum_{t=-\infty}^{\infty} \chi[t]e^{-i\omega t}$$

对于傅里叶分析来说，最高效的算法非快速傅里叶变换（Fast Fourier Transform，FFT）莫属。这个算法已经在 SciPy 与 NumPy 这两个库中实现了。

当应用于时间序列数据时，傅里叶分析能够将数据从时域映射到频域上面，从而得到一个频谱。对于某些频谱来说，它们会在频谱的特定频率上表现为一些尖峰。例如，乐谱就是由代表不同频率的音符构成的，其中 A 音符的声高为 440Hz。实际上，A 音符代表的就是敲击音叉时所产生的声音。借助钢琴之类的乐器，我们不仅可以演奏出 A 音符，还可以演奏出其他音符。白噪声是由许多不同频率的信号构成的，而且这些信号的功率一样。白光也是由不同频率的所有可见光混合而成的，这些可见光的功率也是一样的。

在下面的代码中，我们要导入两个函数（具体参阅 ch-07.ipynb 文件）。

```
from scipy.fftpack import rfft
from scipy.fftpack import fftshift
```

这里，rfft()函数可以对实值数据进行 FFT。此外，我们还可以使用 FFT()函数，但是它用于实值数据集时，会发出警告。函数 fftshift()可以把零频分量（数据的平均值）移动到频谱中央，这样看起来会更舒服一些。为了便于理解，下面我们以正弦波为例进行讲解。首先，我们创建一个正弦波，然后对其实施 FFT，具体代码如下。

```
t = np.linspace(-2 * np.pi, 2 * np.pi, len(sunspots))
mid = np.ptp(sunspots)/2
sine = mid + mid * np.sin(np.sin(t))

sine_fft = np.abs(fftshift(rfft(sine)))
print "Index of max sine FFT", np.argsort(sine_fft)[-5:]
```

下面输出的是对应于最大振幅的相应索引。

```
Index of max sine FFT [160 157 166 158 154]
```

我们对太阳黑子数据进行 FFT。

```
transformed = np.abs(fftshift(rfft(sunspots)))
print "Indices of max sunspots FFT", np.argsort(transformed)[-5:]
```

通过下列索引我们可以看出，频谱中有 5 个大峰值。

```
Indices of max sunspots FFT [205 212 215 209 154]
```

我们看到，这里 154 处也有一个最大峰值。最终结果如图 7-8 所示。

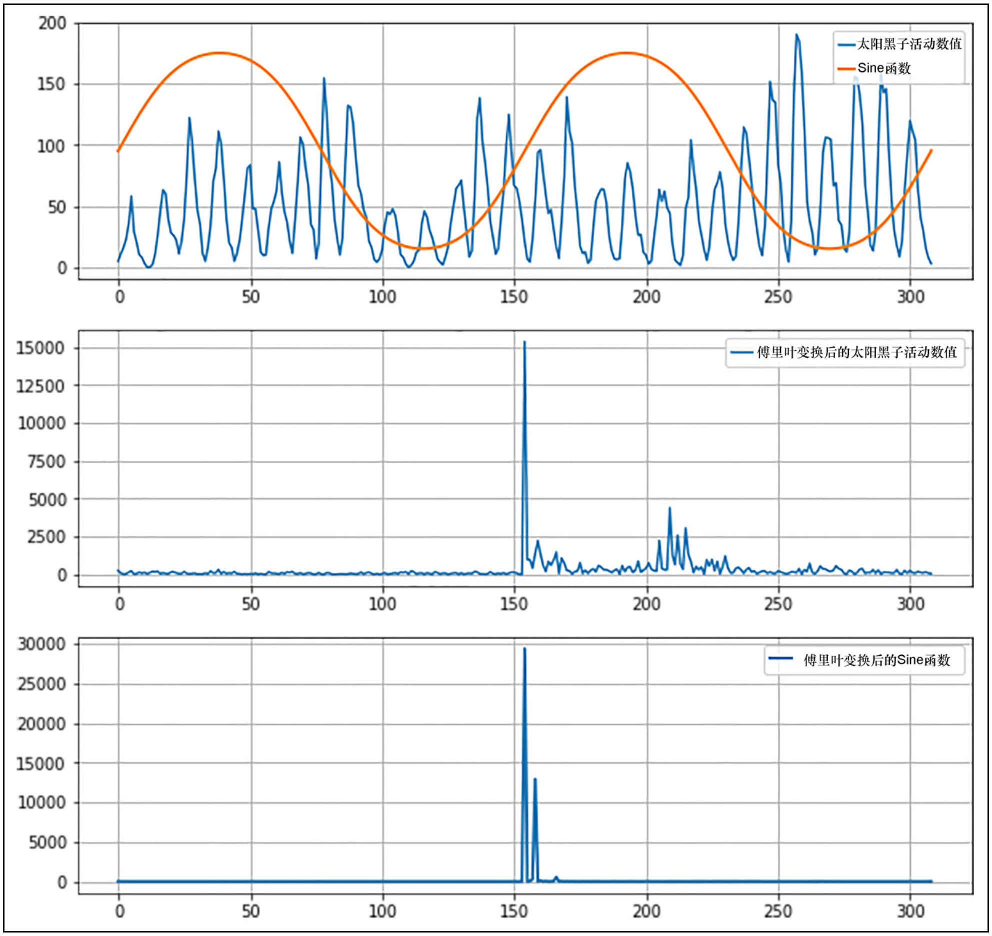


图 7-8

下面的代码摘自本书代码包中的 ch-07.ipynb 文件。

```
import numpy as np
import statsmodels.api as sm
import matplotlib.pyplot as plt
from scipy.fftpack import rfft
from scipy.fftpack import fftshift
data_loader = sm.datasets.sunspots.load_pandas()
sunspots = data_loader.data["SUNACTIVITY"].values

t = np.linspace(-2 * np.pi, 2 * np.pi, len(sunspots))
mid = np.ptp(sunspots)/2
```

```
sine = mid + mid * np.sin(np.sin(t))

sine_fft = np.abs(fftshift(rfft(sine)))
print("Index of max sine FFT", np.argsort(sine_fft)[-5:])

transformed = np.abs(fftshift(rfft(sunspots)))
print("Indices of max sunspots FFT", np.argsort(transformed)[-5:])

plt.subplot(311)
plt.plot(sunspots, label="Sunspots")
plt.plot(sine, lw=2, label="Sine")
plt.grid(True)
plt.legend()
plt.subplot(312)
plt.plot(transformed, label="Transformed Sunspots")
plt.grid(True)
plt.legend()
plt.subplot(313)
plt.plot(sine_fft, lw=2, label="Transformed Sine")
plt.grid(True)
plt.legend()
plt.show()
```

7.10　谱分析

在 7.9 节中，我们绘制了数据集的振幅频谱。而物理信号的功率频谱可以直观展现出该信号的能量分布。对于前面的代码，我们稍作修改就能用来绘制功率频谱。具体做法是将某些值取平方，代码如下。

```
plt.plot(transformed ** 2, label="Power Spectrum")
```

相位谱可以为我们直观展示相位，即正弦函数的起始角，相应代码如下。

```
plt.plot(np.angle(transformed), label="Phase Spectrum")
```

最终结果如图 7-9 所示。

至于完整的代码，请读者参阅本书代码包中的 ch-07.ipynb 文件。

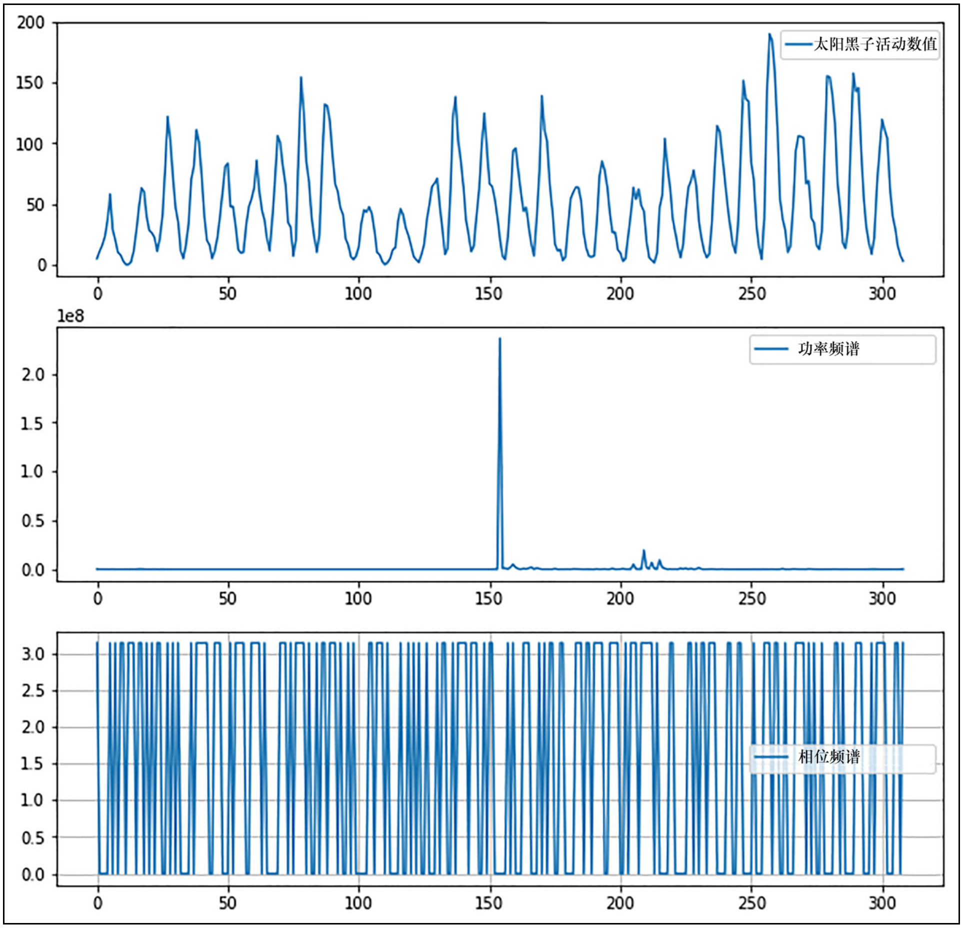


图 7-9

7.11 滤波

滤波是一种信号处理技术，可以对信号的某些部分进行删减或抑制。应用 FFT 后，我们就可以对高频或低频进行过滤或者设法删除白噪声了。白噪声是功率频谱为常数的一个随机信号，因此，它不包含任何有用信息。scipy.Signal 程序包为滤波提供了许多相应的实用程序。下面的例子将展示部分函数的使用方法。

- 中值滤波器（Median Filter）可以用来计算滚动窗口中数据的中值，这个滤波器是由 medfilt()函数实现的，我们可以通过可选参数来指定窗口大小。

- Wiener 滤波器能够通过统计数值来删除噪音，对于一个滤波器 g(t)与一个信号 s(t)，我们可以通过 (g * [s + n])(t)来计算其卷积。这个滤波器是通过 wiener()实现的，它同样也有一个指定窗口大小的可选参数。
- detrend 滤波器可以用来删除趋势。它可以是一个线性或者不变趋势。这个滤波器是由 detrend()函数实现的。

下列代码摘自本书代码包中的 ch-07.ipynb 文件。

```
import statsmodels.api as sm
import matplotlib.pyplot as plt
from scipy.signal import medfilt
from scipy.signal import wiener
from scipy.signal import detrend

data_loader = sm.datasets.sunspots.load_pandas()
sunspots = data_loader.data["SUNACTIVITY"].values
years = data_loader.data["YEAR"].values

plt.plot(years, sunspots, label="SUNACTIVITY")
plt.plot(years, medfilt(sunspots, 11), lw=2, label="Median")
plt.plot(years, wiener(sunspots, 11), '--', lw=2, label="Wiener")
plt.plot(years, detrend(sunspots), lw=3, label="Detrend")
plt.xlabel("YEAR")
plt.grid(True)
plt.legend()
plt.show()
```

最终结果如图 7-10 所示。

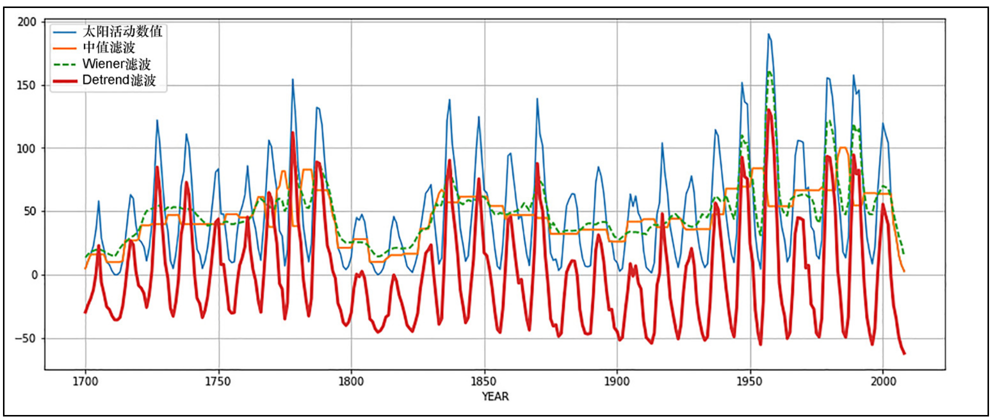


图 7-10

7.12 小结

本章的时间序列示例使用了年度太阳黑子周期数据。

我们介绍了一个比较常见的问题的解决方法，即在同一个时间序列中找出一个值与固定周期数之前的另一个数据点或者一组数据点之间的关系。

对于移动平均法，需要指定一个窗口来明确规定向前可以看到的数据，此后，窗口每一次前移一个周期，都要计算一次窗口内数据的平均值。在 Pandas 的应用程序接口中，rolling_window()函数为我们提供了许多窗口函数的功能，其中字符串参数 win_type 取不同的值，就对应不同的窗口函数。

协整类似于相关性，是一个度量两个时间序列之间的关系的指标。进行回归分析时，我们经常遇到过拟合的问题。这个问题表现为，模型对于样本拟合的效果非常理想，但是当引入新数据点后，效果却变得很糟糕。为了对模型进行评估，我们可以求取适当的评价指标。

对于数据分析而言，数据库是一个非常重要的工具。自从 20 世纪 70 年代开始，关系型数据库便开始流行起来。现在，NoSQL 数据库已经变成一个可行的替代方案。第 8 章将会介绍各种相关的数据库（包括关系型与 NoSQL 数据库）及其应用程序接口。

第 8 章 应用数据库

在本章中，我们将为读者介绍各种数据库及其应用编程接口。这里所说的数据库包括关系型数据库以及非关系型（NoSQL）数据库。关系型数据库是由数据表汇集而成的，更重要的是，这些数据表中的数据是按照数据项之间的关系进行组织的。当然，这里所说的关系，也可以是某个数据表中的行数据与其他数据表中的行数据之间的关系。关系型数据库不仅涉及数据表之间的关系，首先，它要处理同一个数据表中不同列之间的关系（很明显，一个数据表内的各列毫无疑问是相关的）；其次，它还要处理数据表之间的关系。

伴随着大数据和 Web 应用的流行，非关系型（Not Only SQL，NoSQL）数据库也开始野蛮生长。NoSQL 系统将成为类 SQL 事实上的标准。NoSQL 数据库的主旨在于，使用比关系模型更为灵活的方式来存储数据。这就可能意味着，无需数据库模式或者灵活的数据库模式。当然，灵活性和速度也是有代价的，例如无法始终保证事务的一致性。NoSQL 数据库可以利用面向列的方法以字典的形式来储存数据，这些数据对象包括文档、对象、图、元组，甚至这些对象的组合体。本章将要介绍的主题如下。

- 基于 sqlite3 的轻量级访问
- 通过 Pandas 访问数据库
- SQLAlchemy
- Pony ORM
- Dataset：懒人数据库
- PyMongo 与 MongoDB
- 利用 Redis 存储数据
- 利用 memcache 存储数据

- Apache Cassandra。

8.1　基于 sqlite3 的轻量级访问

SQLite 是一款非常流行的关系型数据库，它非常轻盈，因此被大量应用程序广泛采纳，如 Mozilla Firefox 等浏览器。安卓系统上的大部分应用的数据存储也是通过 SQLite 完成的。

sqlite3 是 Python 标准发行版自带的一个模块，可以用于处理 SQLite 数据库。使用 sqlite3 模块时，数据库既可以存放到文件中，也可以保留在内存中。本例采用后一种方式。下面我们导入 sqlite3，代码如下。

```
import sqlite3
```

首先，连接数据库。如果希望把数据库存到文件中，那么必须提供一个文件名，否则，可以通过下列方式将数据库留在内存中。

```
with sqlite3.connect(":memory:") as con:
```

上面使用了 Python 的 with 语句。需要注意的是，这个语句依赖于特定上下文管理器类的__exit__()方法的存在。如果使用了这个语句，我们就无需显式关闭数据库连接了。这是因为上下文管理器会自动替我们关闭连接。连接到数据库后，我们还需要一个游标。游标在数据库中的作用，至少从概念上来讲，类似于文本编辑器中的光标。注意，这个游标也需要由我们来关闭。

下面开始创建游标，代码如下。

```
c = con.cursor()
```

现在，可以直接创建数据表了。通常，必须首先创建一个数据库，或数据库专家已经替我们建立了一个数据库。在本章中，我们不仅需要了解 Python，而且还得懂 SQL。SQL 是一种专门用来查询和操作数据库的语言。限于篇幅，这里不可能对 SQL 进行详尽地介绍。不过，只要稍微努力，读者就能够掌握基本的 SQL。为了创建数据表，我们需要向游标传递一个 SQL 字符串，具体如下。

```
c.execute('''CREATE TABLE sensors (date text, city text,
code text, sensor_id real, temperature real)''')
```

上面的代码会创建一个包含很多列的数据表，具体名称为 sensors。在上面的字符串中，

text 和 real 用来表明字符串和数值的类型。通过上面的代码，我们就能创建一个可用的数据表了。如果创建过程发生错误，我们就会收到相应的出错提示。列出数据库中数据表的方法与数据库本身紧密相关。通常，有一个或一组专门的数据表来存放用户数据表的元数据。下面我们列出 SQLite 数据表，具体如下。

```
for table in c.execute("SELECT name FROM sqlite_master
WHERE type = 'table'"):
print("Table", table[0])
```

正如所料，我们将得到如下的输出。

```
Table sensors
```

现在，我们要插入并查询一些随机数据，具体如下。

```
c.execute("INSERT INTO sensors VALUES ('2016-11-05',
'Utrecht','Red',42,15.14)")
c.execute("SELECT * FROM sensors")
print c.fetchone()
```

下面我们输出插入的记录。

```
(u'2016-11-05', u'Utrecht', u'Red', 42.0, 15.14)
```

当不再需要某个数据表时，我们就可以将其删除了。需要注意的是，删除是一项非常危险的操作，因此，我们必须绝对肯定再也用不到这个数据表了。因为数据表一旦被删，就无法恢复了，除非之前已经做好了备份。下面的代码将删除数据表并显示删除操作执行后所剩数据表的数量，具体如下。

```
con.execute("DROP TABLE sensors")
print ("# of tables", c.execute("SELECT COUNT(*)
FROM sqlite_master WHERE type = 'table'").fetchone()[0])
```

输出内容如下。

```
# of tables 0
```

下列代码摘自本书代码包中的 ch-08.ipynb 文件。

```
import sqlite3

with sqlite3.connect(":memory:") as con:
```

```
    c = con.cursor()
    c.execute('''CREATE TABLE sensors(date text, city text,
code text, sensor_id real, temperature real)''')

    for table in c.execute("SELECT name FROM sqlite_master
WHERE type = 'table'"):
        print("Table", table[0])

    c.execute("INSERT INTO sensors VALUES ('2016-11-05',
'Utrecht','Red',42,15.14)")
    c.execute("SELECT * FROM sensors")
    print(c.fetchone())
    con.execute("DROP TABLE sensors")

    print("# of tables", c.execute("SELECT COUNT(*)
FROM sqlite_master WHERE type = 'table'").fetchone()[0])

    c.close()
```

8.2 通过 Pandas 访问数据库

我们可以向 Pandas 提供一个数据库连接，如前面例子中的连接或 SQLAlchemy 连接。关于 SQLAlchemy 连接，我们将在本章后面部分进行讲解。正如第 7 章所述，这里也要用到 statsmodels 模块的太阳黑子活动数据。为了加载数据，我们可以使用下列代码。

（1）创建元组列表，以构建 Pandas DataFrame。

```
rows = [tuple(x) for x in df.values]
```

与之前的示例不同，这里要创建的是一个未规定数据类型的数据表，代码如下。

```
con.execute("CREATE TABLE sunspots(year, sunactivity)")
```

（2）我们知道，executemany()方法可以执行多条语句。就本例而言，我们要插入一些记录，这些记录来自元组列表。下面将这些数据行插入数据表并显示行数。

```
con.executemany("INSERT INTO sunspots(year, sunactivity) VALUES
(?, ?)", rows)
c.execute("SELECT COUNT(*) FROM sunspots")
print(c.fetchone())
```

数据表的行数如下。

```
(309,)
```

（3）execute()函数返回结果中 rowcount 属性存放的是受影响的数据行的数量。这个属性有点古怪，而且与 SQLite 版本密切相关。另外，就像上面代码中看到的那样，SQL 查询是无歧义的。下面我们删除事件数大于 20 的记录，代码如下。

```
print("Deleted", con.execute("DELETE FROM sunspots where
sunactivity > 20").rowcount, "rows")
```

结果如下。

```
Deleted 217 rows
```

（4）如果把数据库连接至 Pandas，就可以利用 read_sql()函数来执行查询并返回 Pandas DataFrame 了。下面选择前 1732 条记录，代码如下。

```
print(read_sql("SELECT * FROM sunspots where year < 1732", con))
```

最终得到如下 Pandas DataFrame。

```
    year  sunactivity
0   1700            5
1   1701           11
2   1702           16
3   1707           20
4   1708           10
5   1709            8
6   1710            3
7   1711            0
8   1712            0
9   1713            2
10  1714           11
11  1723           11

[12 rows x 2 columns]
```

以下代码摘自本书代码包中的 ch-08.ipynb 文件。

```
import statsmodels.api as sm
from pandas.io.sql import read_sql
import sqlite3

with sqlite3.connect(":memory:") as con:
    c = con.cursor()
```

```
    data_loader = sm.datasets.sunspots.load_pandas()
    df = data_loader.data
    rows = [tuple(x) for x in df.values]

    con.execute("CREATE TABLE sunspots(year, sunactivity)")
    con.executemany("INSERT INTO sunspots(year, sunactivity) VALUES (?,
?)", rows)
    c.execute("SELECT COUNT(*) FROM sunspots")
    print(c.fetchone())
    print("Deleted", con.execute("DELETE FROM sunspots where sunactivity >
20").rowcount, "rows")
    print(read_sql("SELECT * FROM sunspots where year < 1732", con))
    con.execute("DROP TABLE sunspots")

    c.close()
```

8.3 SQLAlchemy

SQLAlchemy 以基于设计模式（design pattern）的对象关系映射（ORM）而闻名。也就是说，它可以把 Python 的类映射为数据库的数据表。实际上，这意味着添加了一个额外的抽象层，因此，我们需要使用 SQLAlchemy 应用程序接口来跟数据库打交道，而非使用 SQL 命令。使用 SQLAlchemy 的好处是，它能够在幕后替我们处理各种细节。不过，凡事有利皆有弊，使用 SQLAlchemy 的缺点是我们不得不学习其应用程序接口，同时，性能也会有所下降。本节将学习如何安装 SQLAlchemy 以及如何通过 SQLAlchemy 填充和查询数据库。

8.3.1 SQLAlchemy 的安装和配置

下面是安装 SQLAlchemy 所需的命令。

```
$ pip3 install sqlalchemy
```

在编写本书时，SQLAlchemy 的最新版本是 1.1.4。从其官网可以找到 SQLAlchemy 的安装程序和代码仓库。

此外，SQLAlchemy 还有一个支持页面，在使用 SQLAlchemy 的时候，需要定义一个超类，代码如下。

```
from sqlalchemy.ext.declarative import declarative_base
Base = declarative_base()
```

本节及后面几节中会使用一个带有两个数据表的小型数据库，其中第 1 个数据表是关于观测站的，第 2 个数据表是描述观测站内的传感器的。每个观测站可以有 0 个、1 个或者多个传感器，其中，每个观测站都有一个整数 ID，这些数字是由数据库自动生成的。此外，每个观测站都有一个名称，而且这个名称是唯一的，也是强制性的。

同时，每个传感器也有自己的整数 ID。我们将记录传感器的最后一次观察值。这个值可以是观测值的倍数。具体情况读者可以参考本书代码包中的 ch-08.ipynb 代码。注意，我们不必直接运行这个脚本，因为它是供其他脚本使用的。

```
from sqlalchemy import Column, ForeignKey, Integer, String, Float
from sqlalchemy.ext.declarative import declarative_base
from sqlalchemy.orm import relationship
from sqlalchemy import create_engine
from sqlalchemy import UniqueConstraint

Base = declarative_base()
class Station(Base):
    __tablename__ = 'station'
    id = Column(Integer, primary_key=True)
    name = Column(String(14), nullable=False, unique=True)

    def __repr__(self):
        return "Id=%d name=%s" %(self.id, self.name)

class Sensor(Base):
    __tablename__ = 'sensor'
    id = Column(Integer, primary_key=True)
    last = Column(Integer)
    multiplier = Column(Float)
    station_id = Column(Integer, ForeignKey('station.id'))
    station = relationship(Station)

    def __repr__(self):
        return "Id={:d} last={:d} multiplier={:.1f} station_id=
        {:d}".format(
        self.id,self.last, self.multiplier, self.station_id)

if __name__ == "__main__":
    print("This script is used by code further down in this notebook.")
```

8.3.2　通过 SQLAlchemy 填充数据库

数据表的创建将在后面介绍。本节先来准备一个脚本，以便用来填充数据库（注意，

不用直接运行这个脚本，因为它是供 8.3.3 节中的脚本使用的）。通过 DBSession 对象，可以向数据表中插入数据。当然，我们还需要一个引擎，不过，引擎的创建方法也留到 8.3.3 节介绍。

1．创建 DBSession 对象，代码如下。

```
Base.metadata.bind = engine

DBSession = sessionmaker(bind=engine)
session = DBSession()
```

2．创建两个观测站。

```
de_bilt = Station(name='De Bilt')
session.add(de_bilt)
session.add(Station(name='Utrecht'))
session.commit()
print("Station", de_bilt)
```

在提交这个会话前，这些数据行是不会被插入的。下面是与第一个观测站有关的输出。

```
Station Id=1 name=De Bilt
```

3．同样，还要插入传感器记录，代码如下。

```
temp_sensor = Sensor(last=20, multiplier=.1, station=de_bilt)
session.add(temp_sensor)
session.commit()
print("Sensor", temp_sensor)
```

这个传感器位于第一个观察站中，所以，将得到如下输出内容。

```
Sensor Id=1 last=20 multiplier=0.1 station_id=1
```

数据库填充代码可以在本书代码包中的 ch-08.ipynb 文件中找到。同样，这个脚本也无需直接运行，它也是供其他脚本使用的。填充代码如下。

```
from sqlalchemy import create_engine
from sqlalchemy.orm import sessionmaker

from alchemy_entities import Base, Sensor, Station

def populate(engine):
```

```
    Base.metadata.bind = engine

    DBSession = sessionmaker(bind=engine)
    session = DBSession()

    de_bilt = Station(name='De Bilt')
    session.add(de_bilt)
    session.add(Station(name='Utrecht'))
    session.commit()
    print("Station", de_bilt)

    temp_sensor = Sensor(last=20, multiplier=.1, station=de_bilt)
    session.add(temp_sensor)
    session.commit()
    print("Sensor", temp_sensor)

if __name__ == "__main__":
    print("This script is used by code further down in this notebook")
```

8.3.3 通过 SQLAlchemy 查询数据库

下面通过 URI 来创建引擎，代码如下。

```
engine = create_engine('sqlite:///demo.db')
```

通过这个 URI，我们规定要使用的引擎为 SQLite，同时指出要把数据存放到文件 demo.db 中。然后，利用刚才创建的引擎来创建两个数据表，即 station 和 sensor。相应代码如下。

```
Base.metadata.create_all(engine)
```

对于 SQLAlchemy 查询，我们也需要一个 DBSession 对象，这个对象在 8.3.2 节已经介绍过了。

下面选择数据表 station 中的第一行数据。

```
station = session.query(Station).first()
```

下面代码用于选择所有观测站，具体如下。

```
print "Query 1", session.query(Station).all()
```

下面是输出结果。

```
Query 1 [Id=1 name=De Bilt, Id=2 name=Utrecht]
```

选择所有传感器，代码如下。

```
print("Query 2", session.query(Sensor).all())
```

下面是输出结果。

```
Query 2 [Id=1 last=20 multiplier=0.1 station_id=1]
```

下面，选择第一个观测站的第一个传感器，代码如下。

```
print("Query 3",session.query(Sensor).filter(Sensor.station ==
station).one())
```

下面是输出结果。

```
Query 3 Id=1 last=20 multiplier=0.1 station_id=1
```

还可以使用 pandas 的 read_sql()方法进行查询，具体如下。

```
print(read_sql("SELECT * FROM station", engine.raw_connection())
```

输出内容如下。

```
    id    name
0    1  De Bilt
1    2  Utrecht
[2  rows x 2 columns]
```

以下代码摘自本书代码包中的 ch-08.ipynb 文件。

```
from alchemy_entities import Base, Sensor, Station
from populate_db import populate
from sqlalchemy import create_engine
from sqlalchemy.orm import sessionmaker
import os
from pandas.io.sql import read_sql


engine = create_engine('sqlite:///demo.db')
```

```
Base.metadata.create_all(engine)
populate(engine)
Base.metadata.bind = engine
DBSession = sessionmaker()
DBSession.bind = engine
session = DBSession()

station = session.query(Station).first()

print("Query 1", session.query(Station).all())
print("Query 2", session.query(Sensor).all())
print("Query 3", session.query(Sensor).filter(Sensor.station ==
station).one())
print(read_sql("SELECT * FROM station", engine.raw_connection()))

try:
    os.remove('demo.db')
    print("Deleted demo.db")
except OSError:
    pass
```

8.4 Pony ORM

Pony ORM 是 Python 编程语言下的另一款 ORM 程序包。Pony ORM 是用纯 Python 编写的，它能自动进行查询优化，同时提供了一个图形用户界面的数据库模式编辑器。此外，它还支持自动事务处理、自动缓存和组合关键字（Composite Keys）。有了 Pony ORM，我们就可以通过 Python 的生成器表达式来查询数据库了。当然，这些生成器最终都会转换为 SQL。

安装这个程序库的命令如下。

```
$ pip3 install pony
```

本例中我们将用到这个程序包，因此需要将其导入。下面的导入代码摘自本书代码包中的 pony_ride.py 文件。

```
from pony.orm import Database, db_session
import statsmodels.api as sm
```

下面我们来创建一个 in-memory 型的 SQLite 数据库。

```
db = Database('sqlite', ':memory:')
```

下面通过 Pandas 的 write_frame()函数来加载太阳黑子数据并将其写入数据库。具体代码如下。

```
with db_session:
    data_loader = sm.datasets.sunspots.load_pandas()
    df = data_loader.data
    df.to_sql("sunspots", db.get_connection())
    print(db.select("count(*) FROM sunspots"))
```

这个太阳黑子数据表的行数如下。

```
[309]
```

8.5 Dataset：懒人数据库

Dataset 是一个 Python 库，实际上就是 SQLAlchemy 的一个包装器。这个库的开发主旨是尽量做到易于使用，也就是尽量让懒人满意。

安装 dataset 的命令如下。

```
$ pip3 install dataset
```

创建并连接一个 in-memory 型的 SQLite 数据库，代码如下。

```
import dataset
db = dataset.connect('sqlite:///:memory:')
```

创建一个名为 books 的数据表，代码如下。

```
table = db["books"]
```

事实上，在数据库中这个表尚未创建，因为我们还没有为其指定任何列。我们只是创建了一个相关的对象。我们调用 insert()方法时，会自动创建数据表模式，同时，向 insert()方法传递含有书名的字典，具体代码如下。

```
table.insert(dict(, author='Ivan Idris'))
table.insert(dict(,
author='Ivan Idris'))
table.insert(dict(,
```

```
author='Ivan Idris'))
```

通过下列代码，可以显示数据表各行的内容。

```
for row in db['books']:
   print(row)
```

结果如下。

```
<dataset.persistence.util.ResultIter object at 0x10d4bf550>
OrderedDict([('id', 1), ('title', "NumPy Beginner's Guide"), ('author',
'Ivan Idris')])
OrderedDict([('id', 2), ('title', 'NumPy Cookbook'), ('author', 'Ivan
Idris')])
OrderedDict([('id', 3), ('title', 'Learning NumPy'), ('author', 'Ivan
Idris')])
```

现在，我们可以轻松显示该数据库中各个数据表了，代码如下。

```
print("Tables", db.tables)
```

上述代码的输出结果如下。

```
Tables ['books']
```

下面的代码摘自本书代码包中的 ch-08.ipynb 文件。

```
import dataset

db = dataset.connect('sqlite:///:memory:')
table = db["books"]
table.insert(dict(, author='Ivan Idris'))
table.insert(dict(, author='Ivan Idris'))
table.insert(dict(, author='Ivan Idris'))
for row in db['books']:
   print(row)

print("Tables", db.tables)
```

8.6 PyMongo 与 MongoDB

MongoDB 是一个面向文档的 NoSQL 数据库，其名称取自 humongous 一词，即硕大

无比之意，其中，文档将以类似 JSON 的 BSON 格式进行存储。我们可以下载 MongoDB，安装过程非常简单，我们只需将压缩文件解压即可。创作本书时，这个程序库的版本为 3.4.0。这个版本的 bin 目录下面有一个名为 mongod 的文件，用来启动服务器。MongoDB 位于/data/db 目录下面，该目录就是存储数据的地方。当然，可以通过下列命令来指定其他目录。

```
$ mkdir /tmp/db
```

进入存放数据库二进制可执行代码的目录并启动数据库，命令如下。

```
./mongod --dbpath /tmp/db
```

这个进程需要一直运行，这样我们才能查询数据库。

PyMongo 是 MongoDB 的 Python 驱动程序，这个程序的安装方法如下。

```
$ pip3 install pymongo
```

与 MongoDB 的测试数据库建立连接，代码如下。

```
from pymongo import MongoClient
client = MongoClient()
db = client.test_database
```

别忘了，我们是可以利用 pandas DataFrame 来创建 JSON 的。下面创建 JSON 并将其存于 MongoDB 中，代码如下。

```
data_loader = sm.datasets.sunspots.load_pandas()
df = data_loader.data
rows = json.loads(df.T.to_json()).values()
db.sunspots.insert(rows)
```

现在，我们来查询刚才创建的文档。

```
cursor = db['sunspots'].find({})
df = pd.DataFrame(list(cursor))
print(df)
```

这将打印输出整个 pandas DataFrame。下列代码摘自本书代码包中的 ch-08.ipynb 文件。

```
from pymongo import MongoClient
import statsmodels.api as sm
```

```
import json
import pandas as pd

client = MongoClient()
db = client.test_database

data_loader = sm.datasets.sunspots.load_pandas()
df = data_loader.data
rows = json.loads(df.T.to_json()).values()
db.sunspots.insert_many(rows)

cursor = db['sunspots'].find({})
df = pd.DataFrame(list(cursor))
print(df)

db.drop_collection('sunspots')
```

8.7 利用 Redis 存储数据

Redis 是一个 in-memory 型的键-值数据库，由 C 语言编写而成。Redis 这个名称源自 REmote DIctionary Server，即远程字典服务器。处于内存存储模式时，Redis 的速度快得惊人，而且读写操作的速度几乎一样快。Redis 遵循发布订阅模式并且通过 Lua 脚本来处理存储过程。发布订阅模式通过客户端可订阅的信道来接收消息。创作本书时的 Redis 最新版本为 3.2.6，我们可以下载 Redis。完成安装之后，我们就可以通过以下命令来运行服务器了。

```
$ src/redis-server
```

下面开始安装一个 Python 驱动程序，命令如下。

```
$ pip3 install redis
```

Redis 的使用非常简便，只要把它当作一个庞大的字典即可。不过，Redis 也有自身的局限性。有时，它的易用性仅仅体现在将复杂的对象存储为 JSON 字符串或其他格式。这正是我们联合使用 Pandas DataFrame 的原因。下面，我们与 Redis 建立连接，代码如下。

```
r = redis.StrictRedis()
```

通过 JSON 字符串创建一个键-值对，代码如下。

```
r.set('sunspots', data)
```

通过下列代码检索数据。

```
blob = r.get('sunspots')
```

下面的代码非常简单，它们摘自本书代码包中的 ch-08.ipynb 文件。

```
import redis
import statsmodels.api as sm
import pandas as pd

r = redis.StrictRedis()
data_loader = sm.datasets.sunspots.load_pandas()
df = data_loader.data
data = df.T.to_json()
r.set('sunspots', data)
blob = r.get('sunspots')
print(pd.read_json(blob))
```

8.8 利用 memcache 存储数据

与 Redis 类似，memcache 也是一个 in-memory 型的键-值数据库。在安装运行 memcache 服务器之后，就可以通过下列命令来安装 memcache 的 Python 客户端了。

```
$ pip3 install python3-memcache
```

下面的代码摘自 ch-08.ipynb 文件，它将创建一个 memcache 客户端，然后将 DataFrame 存储到 Memcache，其中自动过期值为 600 秒。这段代码与前面介绍 Redis 时看到的代码非常相似。

```
import memcache
import statsmodels.api as sm
import pandas as pd

client = memcache.Client([('127.0.0.1', 11211)])
data_loader = sm.datasets.sunspots.load_pandas()
df = data_loader.data
data = df.T.to_json()
client.set('sunspots', data, time=600)
print("Stored data to memcached, auto-expire after 600 seconds")
blob = client.get('sunspots')
```

```
print(pd.read_json(blob))
```

8.9　Apache Cassandra

Apache Cassandra 是结合了键-值和传统关系型数据库特性的混合型数据库。对于传统的关系型数据库而言，数据表中的列是固定的。可是，对于 Cassandra 来说，同一个数据表中的各行可以具有不同的列。从这个角度看，Cassandra 是一种面向列的数据库，因为它允许各行灵活使用不同的模式。这个数据库中的各列，都是按照所谓的列族（Column Family）进行组织的，这里的列族相当于关系型数据库中的数据表。Cassandra 数据库已经摒弃了各种连接和子查询操作。读者可以自行下载 Cassandra。在本书写作时，这个数据库的最新版本是 3.9。

（1）从命令行运行服务器，命令如下。

```
$ bin/cassandra-f
```

（2）可以创建 conf/cassandra.yaml 中列出的目录，或者进行如下的调整。

```
data_file_directories:
/tmp/lib/cassandra/data
commitlog_directory: /tmp/lib/cassandra/commitlog
saved_caches_directory: /tmp/lib/cassandra/saved_caches
```

（3）如果不想保存某些数据，可以考虑下列命令。

```
$ mkdir -p /tmp/lib/cassandra/data
$ mkdir -p /tmp/lib/cassandra/commitlog
$ mkdir -p /tmp/lib/cassandra/saved_caches
```

（4）下面安装 Python 驱动程序，命令如下。

```
$ pip3 install cassandra-driver
```

（5）下面开始编写代码。首先，与集群建立连接并创建一个会话，代码如下。

```
cluster = Cluster()
session = cluster.connect()
```

（6）在 Cassandra 中，有一个所谓的 keyspace 的概念，它实际上就是用来存放数据表的一个容器。Cassandra 建立了自己的查询语言，名为 Cassandra 查询语言（Cassandra Query

Language，CQL）。CQL 的用法与 SQL 类似。下面创建 keyspace 并设置使用该 keyspace 的会话，代码如下。

```
session.execute("CREATE KEYSPACE IF NOT EXISTS mykeyspace WITH
REPLICATION = { 'class' : 'SimpleStrategy',
'replication_factor' : 1 };")
session.set_keyspace('mykeyspace')
```

（7）现在，创建一个数据表来存放太阳黑子数据，代码如下。

```
session.execute("CREATE TABLE IF NOT EXISTS sunspots (year
decimal PRIMARY KEY, sunactivity decimal);")
```

（8）创建一条语句，该语句将在循环语句中把元组作为数据行来插入，代码如下。

```
query = SimpleStatement(
    "INSERT INTO sunspots (year, sunactivity) VALUES (%s, %s)",
    consistency_level=ConsistencyLevel.QUORUM)
```

（9）下列代码用于插入数据。

```
for row in rows:
    session.execute(query, row)
```

（10）取得数据表中数据的行数。

```
rows=session.execute("SELECT COUNT(*) FROM sunspots")
for row in rows:
    print(row)
```

输出的行数如下。

```
[Row(count=309)]
```

（11）删除 keyspace，关闭集群。

```
session.execute('DROP KEYSPACE mykeyspace')
cluster.shutdown()
```

下列代码摘自本书代码包中的 ch-08.ipynb 文件。

```
from cassandra import ConsistencyLevel
from cassandra.cluster import Cluster
```

```
from cassandra.query import SimpleStatement
import statsmodels.api as sm

cluster = Cluster()
session = cluster.connect()
session.execute("CREATE KEYSPACE IF NOT EXISTS mykeyspace WITH REPLICATION
= { 'class' : 'SimpleStrategy', 'replication_factor' : 1 };")
session.set_keyspace('mykeyspace')
session.execute("CREATE TABLE IF NOT EXISTS sunspots (year decimal PRIMARY
KEY, sunactivity decimal);")

query = SimpleStatement(
    "INSERT INTO sunspots (year, sunactivity) VALUES (%s, %s)",
    consistency_level=ConsistencyLevel.QUORUM)

data_loader = sm.datasets.sunspots.load_pandas()
df = data_loader.data
rows = [tuple(x) for x in df.values]
for row in rows:
    session.execute(query, row)

rows=session.execute("SELECT COUNT(*) FROM sunspots")
for row in rows:
    print(row)

session.execute('DROP KEYSPACE mykeyspace')
cluster.shutdown()
```

8.10 小结

我们可以把年度太阳黑子周期数据存储在不同的数据库中，包括关系型数据库和 NoSQL 数据库。

这里所谓的关系，不仅限于数据表之间的关系，首先，它与一个数据表内部各列之间的关系有关；其次，它还涉及数据表之间的关系。

实际上，我们可以通过 Python 的标准模块 sqlite3 来跟 SQLite 数据库打交道。我们可以通过 pandas 与 SQLite 数据库或 SQLAlchemy 数据库来建立连接。

SQLAlchemy 以其基于设计模式的 ORM 而闻名天下，通过这个库，可以把 Python 的类映射为数据库的数据表。实际上，ORM 模式是一种通用的架构模式，因此同样适用于其

他各种面向对象程序设计语言。SQLAlchemy 将使用数据库的各种技术细节剥离出去，有了它，我们甚至连 SQL 都不用写了。

MongoDB 是一个面向文档的数据仓库，可以存放巨量的数据。

进入 in-memory 模式后，Redis 不仅运行速度极快，而且写操作也几乎与读操作一样快。Redis 是一个键-值型数据仓库，功能上与 Python 的字典相仿。

Apache Cassandra 不仅具有键-值数据库的特性，同时还具备传统的关系型数据库特性。这是一种面向列的数据库，它的各个列都以列族的形式组织在一起，这里的列族相当于关系型数据库中的数据表。在 Apache Cassandra 数据库中，数据行已经摆脱了特定列组合的束缚。

第 9 章将介绍纯文本数据的分析技术，因为纯文本数据在各个组织和互联网上随处可见。一般来说，纯文本数据的非结构化程度都很高，与处理已经清洗并制表的数据相比，分析纯文本数据需要使用一些截然不同的方法。为了分析这类数据，我们需要借助另一个开源 Python 程序包，即 NLTK。NLTK 发展得已经非常完备，而且自身备有相应的数据集。

第 9 章
分析文本数据和社交媒体

在前几章中，我们着重讨论了结构化数据的分析，主要涉及列表格式的数据。实际上，跟结构化数据一样，纯文本也是最常见的格式之一。文本分析需要用到词频分布分析、模式识别、标注、链接和关联分析（link and association analysis）、情感分析和可视化等。这里将借助 Python 自然语言工具包（The Python Natural Language Toolkit，NLTK）来分析文本。NLTK 自身带有一个文本样本集，这个样本集名为 corpora。同时，scikitlearn 程序库也提供了许多文本分析工具，本章也会对这些工具进行简要的介绍。

此外，在本章中还会举例说明网络分析。本章涉及的主题如下。

- 安装 NLTK
- NLTK 简介
- 滤除停用字、姓名和数字
- 词袋模型
- 词频分析
- 朴素贝叶斯分类
- 情感分析
- 创建词云
- 社交网络分析

9.1 安装 NLTK

安装本章所用到的各种库的命令如下。

```
$ pip3 install nltk scikit-learn
```

9.2 NLTK 简介

NLTK 是一个用来分析自然语言文本（如英文句子）的 Python 应用程序接口。NLTK 起源于 2001 年，最初设计是用来进行教学的。

虽然我们在 9.1 节安装了 NLTK，但还不够：还需要下载 NLTK 语料库。需要注意的是，这里的下载量相对较大（约 1.8 GB），但是只需要下载一次。除非我们确切地知道自己需要哪种语料库，否则最好下载所有可用的语料库。从 Python shell 下载语料库的命令如下。

```
$ python3
>>> import nltk
>>> nltk.download()
```

这时，会出现一个 GUI 应用程序，你可以指定要下载的文件以及放到何处，如图 9-1 所示。

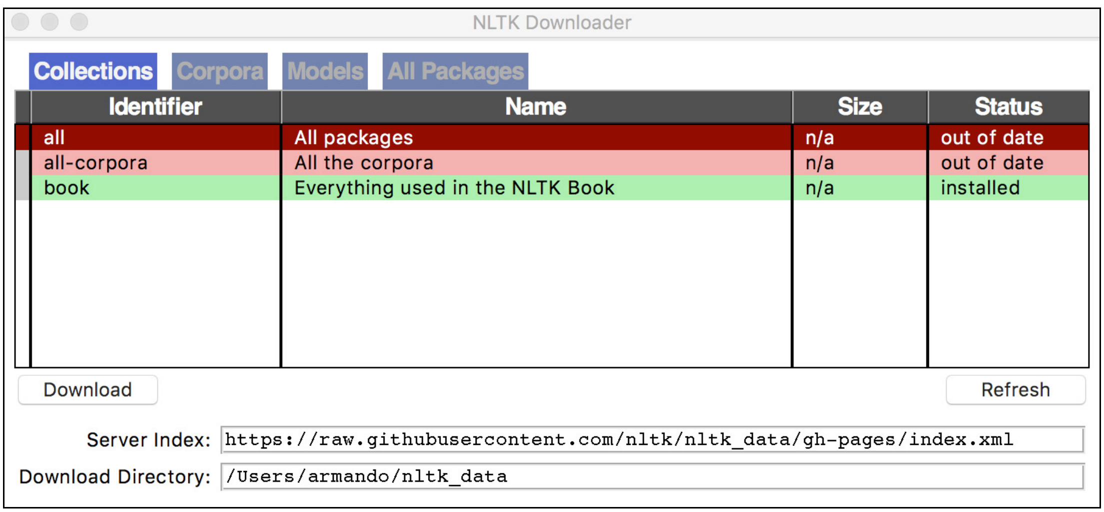


图 9-1

如果是 NLTK 新手，最方便的方法就是选择默认选项并下载所有内容。在本章中，我们将需要用到停用词、电影评论、名字和古腾堡语料库。所以，我们鼓励读者根据 ch-09.ipynb 文件中的代码来学习下列部分。

9.3　滤除停用字、姓名和数字

进行文本分析时，我们经常需要对停用字（Stopwords）进行剔除。这里所谓的停用字，就是那些经常见、但是没有多大信息含量的词。NLTK 为很多语种都提供了停用字语料库。下面加载英语停用字语料并输出部分单词。

```
sw = set(nltk.corpus.stopwords.words('english'))
print("Stop words:", list(sw)[:7])
```

下面是输出的部分常用字。

```
Stop words: ['between', 'who', 'such', 'ourselves', 'an', 'ain', 'ours']
```

请注意，这个语料库中的所有单词都是小写的。

此外，NLTK 还提供了一个 Gutenberg 语料库。Gutenberg 项目是一个数字图书馆计划，供人们在互联网上阅读图书。

下面加载 Gutenberg 语料库并输出部分文件的名称。

```
gb = nltk.corpus.gutenberg
print("Gutenberg files:\n", gb.fileids()[-5:])
```

下面是输出的某些书籍的名称，其中有些你可能比较熟悉。

```
Gutenberg files: ['milton-paradise.txt', 'shakespeare-caesar.txt',
'shakespeare-hamlet.txt', 'shakespeare-macbeth.txt', 'whitman-leaves.txt']
```

现在，从 milton-paradise.txt 文件中提取前两句内容，供下面过滤使用。下面给出具体代码。

```
text_sent = gb.sents("milton-paradise.txt")[:2]
print("Unfiltered:", text_sent)
```

下面是输出的句子。

```
Unfiltered [['[', 'Paradise', 'Lost', 'by', 'John', 'Milton',
'1667', ']'], ['Book', 'I']]
```

现在，过滤掉下面的停用字。

```
for sent in text_sent:
    filtered = [w for w in sent if w.lower() not in sw]
    print("Filtered:\n", filtered)
```

对于第一句，过滤后变成以下格式。

```
Filtered ['[', 'Paradise', 'Lost', 'John', 'Milton', '1667', ']']
```

注意，与之前相比，单词 by 已经被过滤掉了，因为它出现在停用字语料库中了。有时，我们希望将文本中的数字和姓名也删掉。我们可以根据词性（Part of Speech，POS）标签来删除某些单词。在这个标注方案中，数字对应着基数（Cardinal Number，CD）标签。

姓名对应着单数形式的专有名词（the proper noun singular，NNP）标签。标注是基于启发式方法进行处理的，当然，这不可能非常精确。这个主题过于宏大，恐怕需要整本书来进行描述，详见前言部分。在过滤文本时，我们可以使用 pos_tag()函数获取文本内所含的标签，具体如下。

```
tagged = nltk.pos_tag(filtered)
print("Tagged:\n", tagged)
```

对于我们的文本，将得到如下各种标签。

```
Tagged [('[', 'NN'), ('Paradise', 'NNP'), ('Lost', 'NNP'), ('John', 'NNP'),
('Milton', 'NNP'), ('1667', 'CD'), (')', 'CD')]
```

上面的 pos_tag()函数将返回一个元组列表，其中各元组的第二个元素就是文本对应的标签。可见，一些单词虽然被标注为 NNP，但是它们并不是。这里所谓的启发式方法，就是如果单词的第一个字母是大写的，那么就将其标注为 NNP。如果将上面的文本全部转换为小写，那么就会得到不同的结果，这个留给读者自行练习。实际上，我们很容易就能利用 NNP 和 CD 标签来删除列表中的单词，代码如下。

```
words= []
    for word in tagged:
        if word[1] != 'NNP' and word[1] != 'CD':
           words.append(word[0])
```

```
    print(words)
```

下列代码摘自本书代码包中的 ch-09.ipynb 文件。

```
import nltk

sw = set(nltk.corpus.stopwords.words('english'))
print("Stop words:", list(sw)[:7])

gb = nltk.corpus.gutenberg
print("Gutenberg files:\n", gb.fileids()[-5:])

text_sent = gb.sents("milton-paradise.txt")[:2]
print("Unfiltered:", text_sent)

for sent in text_sent:
    filtered = [w for w in sent if w.lower() not in sw]
    print("Filtered:\n", filtered)
    tagged = nltk.pos_tag(filtered)
    print("Tagged:\n", tagged)

    words= []
    for word in tagged:
        if word[1] != 'NNP' and word[1] != 'CD':
           words.append(word[0])

    print("Words:\n",words)
```

9.4 词袋模型

所谓词袋模型，即它认为一篇文档是由其中的词构成的一个集合（即袋子），词与词之间没有顺序以及先后的关系。对于文档中的每个单词，我们都要计算它出现的次数，即单词计数（word counts）。据此，我们可以进行类似垃圾邮件识别之类的统计分析。

如果有一批文档，那么每个唯一字（unique word）都可以视为语料库中的一个特征。这里所谓的“特征”，可以理解为参数或者变量。利用所有的单词计数，我们可以为每个文档建立一个特征向量这里的“向量”一词借用的是其数学含义。如果一个单词存在于语料库中，但是不存在于文档中，那么这个特征的值就为 0。令人惊讶的是，NLTK 至今尚未提供可以方便创建特征向量的实用程序。不过，我们可以借用 Python 机器学习库 scikit-learn 中的 CountVectorizer 类来轻松创建特征向量。第 10 章将对 scikit-learn 进行更

详尽的介绍。

下面从 NLTK 的 Gutenberg 语料库加载两个文本文档，代码如下。

```
hamlet = gb.raw("shakespeare-hamlet.txt")
macbeth = gb.raw("shakespeare-macbeth.txt")
```

现在我们去掉英语停用词并创建特征向量，代码如下。

```
cv = sk.feature_extraction.text.CountVectorizer(stop_words='english')
print("Feature vector:\n", cv.fit_transform([hamlet, macbeth]).toarray())
```

下面是两个文档对应的特征向量。

```
Feature vector:
   [[ 1 0 1 ..., 14 0 1]
    [ 0 1 0 ..., 1 1 0]]
```

下面，我们输出部分特征（唯一字）值。

```
print("Features:\n", cv.get_feature_names()[:5])
```

这些特征是按字母顺序排序的。

```
Features:
  ['1599', '1603', 'abhominably', 'abhorred', 'abide']
```

下列代码摘自本书代码包中的 ch-09.ipynb 文件。

```
import nltk
import sklearn as sk

hamlet = gb.raw("shakespeare-hamlet.txt")
macbeth = gb.raw("shakespeare-macbeth.txt")

cv = sk.feature_extraction.text.CountVectorizer(stop_words='english')

print("Feature vector:\n", cv.fit_transform([hamlet, macbeth]).toarray())
print("Features:\n", cv.get_feature_names()[:5])
```

9.5 词频分析

NLTK 提供的 FreqDist 类可以用来将单词封装成字典并计算给定单词列表中各个单词

出现的次数。下面，我们来加载 Gutenberg 项目中莎士比亚的 Julius Caesar 中的文本。

首先，把停用词和标点符号剔除掉，代码如下。

```
punctuation = set(string.punctuation)
filtered = [w.lower() for w in words if w.lower() not in sw and w.lower()
not in punctuation]
```

然后，创建一个 FreqDist 对象并输出频率最高的键和值，代码如下。

```
fd = nltk.FreqDist(filtered)
print("Words", fd.keys()[:5])
print("Counts", fd.values()[:5])
```

输出的键和值如下。

```
Words ['d', 'caesar', 'brutus', 'bru', 'haue']
Counts [215, 190, 161, 153, 148]
```

很明显，这个列表中的第一个字并非英语单词，因此需要添加一种启发式搜索（Heuristic）方法，只选择最少含有两个字符的单词。对于 NLTK 提供的 FreqDist 类，我们既可以使用类似于字典的访问方法，也可以使用更便利的其他方法。下面，我们要提取最常出现的单词以及对应的出现次数。

```
print("Max", fd.max())
print("Count", fd['d'])
```

对应下面的结果，你肯定不会感到吃惊。

```
Max d
Count 215
```

目前，我们只是针对单个单词构成的词汇进行了分析，不过可以推而广之，将分析扩展到含有两个单词和 3 个单词的词汇上，对于后面这两种，我们分别称之为双字词和三字词。对于这两种分析，分别有对应的函数，即 bigrams()函数和 trigrams()函数。现在，我们再次进行文本分析，不过，这次针对的是双字词。

```
fd = nltk.FreqDist(nltk.bigrams(filtered))
print("Bigrams", fd.keys()[:5])
print("Counts", fd.values()[:5])
print("Bigram Max", fd.max())
print("Bigram count", fd[('let', 'vs')])
```

结果如下。

```
Bigrams [('let', 'vs'), ('wee', 'l'), ('mark', 'antony'), ('marke',
'antony'), ('st', 'thou')]
    Counts [16, 15, 13, 12, 12]
    Bigram Max ('let', 'vs')
    Bigram count 16
```

下列代码摘自本书代码包中的 ch-09.ipynb 文件。

```
import nltk
import string

gb = nltk.corpus.gutenberg
words = gb.words("shakespeare-caesar.txt")

sw = set(nltk.corpus.stopwords.words('english'))
punctuation = set(string.punctuation)
filtered = [w.lower() for w in words if w.lower() not in sw and w.lower()
not in punctuation]
fd = nltk.FreqDist(filtered)
print("Words", fd.keys()[:5])
print("Counts", fd.values()[:5])
print("Max", fd.max())
print("Count", fd['d'])

fd = nltk.FreqDist(nltk.bigrams(filtered))
print("Bigrams", fd.keys()[:5])
print("Counts", fd.values()[:5])
print("Bigram Max", fd.max())
print("Bigram count", fd[('let', 'vs')])
```

9.6 朴素贝叶斯分类

分类算法是机器学习算法中的一种，用来判断给定数据项所属的类别，即种类或类型。比如，我们可以根据某些特征来分辨一部电影属于哪个流派等。这样，流派就是我们要预测的类别。第 10 章还会对机器学习做进一步介绍。此刻，我们要讨论的是一个名为朴素贝叶斯分类的流行算法，它常常用于进行文本文档的分析。

朴素贝叶斯分类是一个概率算法，它基于概率与数理统计中的贝叶斯定理。贝叶斯定理给出了利用新证据修正某事件发生的概率的方法。例如，假设一个袋子里装有一些巧克力和其他物品，

但是这些我们没法看到。这时，我们可以用 P(D)表示从袋子中掏出一块深色巧克力的概率。同时，我们用 P(C)代表掏出一块巧克力的概率。当然，因为全概率是 1，所以 P(D)和 P(C)的最大取值也只能是 1。贝叶斯定理指出，后验概率与先验概率和相似度的乘积成正比，具体公式如下。

$$P(D|C)$$

在上面的公式中，P(D|C)是在事件 C 发生的情况下事件 D 发生的可能性。在我们还没有掏出任何物品之前，P(D) = 0.5，因为我们尚未获得任何信息。实际应用这个公式时，我们必须知道 P(C|D)和 P(C)，或者能够间接求出这两个概率。

朴素贝叶斯分类之所以称为朴素，是因为它简单假设特征之间是相互独立的。实践中，朴素贝叶斯分类的效果通常都会很好，说明这个假设得到了一定程度的保证。近来，人们发现这个假设之所以有意义，理论上是有依据的。不过，由于机器学习领域发展迅猛，现在已经发明了多种效果更佳的算法。

下面，我们将利用停用词或标点符号对单词进行分类。这里，我们将字长作为一个特征，因为停用词和标点符号往往都比较短。

为此，我们需要定义如下所示的函数。

```
def word_features(word):
   return {'len': len(word)}

def isStopword(word):
   return word in sw or word in punctuation
```

下面，我们对取自古登堡项目的 shakespeare-caesar.txt 中的单词进行标注，以区分是否为停用词，代码如下。

```
labeled_words = ([(word.lower(), isStopword(word.lower())) for word in
words])
random.seed(42)
random.shuffle(labeled_words)
print(labeled_words[:5])
```

下面显示了 5 个标注后的单词。

```
    [('was', True), ('greeke', False), ('cause', False), ('but', True),
('house', False)]
```

对于每个单词，我们可以求出其长度。

```
featuresets = [(word_features(n), word) for (n, word) in labeled_words]
```

前几章我们介绍过拟合以及通过训练数据集和测试数据集的交叉验证来避免这种情况的方法。下面我们将要训练一个朴素贝叶斯分类器，其中 90%的单词用于训练，剩下的 10%用于测试。首先，创建训练数据集和测试数据集并针对数据展开训练，代码如下。

```
cutoff = int(.9 * len(featuresets))
train_set, test_set = featuresets[:cutoff], featuresets[cutoff:]
classifier = nltk.NaiveBayesClassifier.train(train_set)
```

现在，我们拿出一些单词，检查该分类器的效果。

```
classifier = nltk.NaiveBayesClassifier.train(train_set)
print("'behold' class", classifier.classify(word_features('behold')))
print("'the' class", classifier.classify(word_features('the')))
```

幸运的是，这些单词的分类完全正确：

```
'behold' class False
'the' class True
```

然后，根据测试数据集来计算分类器的准确性，代码如下。

```
print("Accuracy", nltk.classify.accuracy(classifier, test_set))
```

这个分类器的准确度非常高，几乎达到 85%。下面来看哪些特征的贡献最大。

```
print(classifier.show_most_informative_features(5))
```

结果显示如图 9-2 所示，在分类过程中字长的作用最大。

```
len = 7              False : True   =     62.7 : 1.0
len = 6              False : True   =     49.1 : 1.0
len = 1               True : False  =     12.0 : 1.0
len = 2               True : False  =     10.7 : 1.0
len = 5              False : True   =     10.4 : 1.0
```

图 9-2

下列代码摘自本书代码包中的 ch-09.ipynb 文件。

```
import nltk
import string
import random
sw = set(nltk.corpus.stopwords.words('english'))
punctuation = set(string.punctuation)
```

```
def word_features(word):
   return {'len': len(word)}

def isStopword(word):
    return word in sw or word in punctuation
gb = nltk.corpus.gutenberg
words = gb.words("shakespeare-caesar.txt")

labeled_words = ([(word.lower(), isStopword(word.lower())) for word in
words])
random.seed(42)
random.shuffle(labeled_words)
print(labeled_words[:5])

featuresets = [(word_features(n), word) for (n, word) in labeled_words]
cutoff = int(.9 * len(featuresets))
train_set, test_set = featuresets[:cutoff], featuresets[cutoff:]
classifier = nltk.NaiveBayesClassifier.train(train_set)
print("'behold' class", classifier.classify(word_features('behold')))
print("'the' class", classifier.classify(word_features('the')))

print("Accuracy", nltk.classify.accuracy(classifier, test_set))
print(classifier.show_most_informative_features(5))
```

9.7 情感分析

随着社交媒体、产品评论网站及论坛的兴起，用来自动抽取意见的观点挖掘或情感分析也随之变成一个炙手可热的新研究领域。通常情况下，我们希望知道某个意见的性质是正面的、中立的还是负面的。当然，这种类型的分类我们前面就曾遇到过。也就是说，我们有大量的分类算法可用。还有一个方法就是，通过半自动的（经过某些人工编辑）方法来编制一个单词列表，每个单词赋予一个情感分，即一个数值（如单词“good”的情感分为 5，而单词“bad”的情感分为−5）。如果有了这样一张表，就可以给文本文档中的所有单词打分，从而得出一个情感总分。当然，类别的数量可以大于 3，如五星级评级方案。

我们会应用朴素贝叶斯分类方法对 NLTK 的影评语料库进行分析，从而将影评分为正面的或负面的评价。首先，加载影评语料库并过滤掉停用词和标点符号。这些步骤在此略过，因为之前就介绍过了。也可以考虑更精细的过滤方案，不过，需要注意的是，

如果过滤得过火了，就会影响准确性。下面通过 categories()方法对影评文档进行标注，代码如下。

```
labeled_docs = [(list(movie_reviews.words(fid)), cat)
        for cat in movie_reviews.categories()
        for fid in movie_reviews.fileids(cat)]
```

完整的语料库拥有数以万计的唯一字，这些唯一字都可以用作特征，不过，这样做会极大地影响效率。这里，我们选用词频最高的前 5%的单词作为特征。

```
words = FreqDist(filtered)
N = int(.05 * len(words.keys()))
word_features = words.keys()[:N]
```

对于每个文档，可以通过一些方法来提取特征，其中包括以下方法：

- 检查给定文档是否含有某个单词；
- 求出某个单词在给定文档中出现的次数；
- 正则化单词计数，使得正则化后的最大单词计数小于或等于 1；
- 将上面的数值加 1，然后取对数。这里之所以加 1，是为了防止对 0 取对数；
- 利用上面的数值组成一个度量指标。

俗话说“条条大路通罗马”，不过，这其中肯定不乏有些道路能够让我们更安全，也能更快地到达罗马。下面定义一个函数，该函数会使用原始单词计数来作为度量指标，代码为如下。

```
def doc_features(doc):
    doc_words = FreqDist(w for w in doc if not isStopWord(w))
    features = {}
    for word in word_features:
        features['count (%s)' % word] = (doc_words.get(word, 0))
    return features
```

现在，可以训练我们的分类器了，具体方法与前面的例子相似。这里准确性达到 78%，这个成绩已经相当不错了，非常接近情感分析成绩的上限。因为据研究发现，即便是人类，也无法对给定文档的表达出的感情的看法达成一致。因此，要想让情感分析软件获得满分是不可能的事情。

下面的图 9-3 给出包含信息量最大的特征。

```
   count (wonderful) = 2              pos : neg    =     14.7 : 1.0
 count (outstanding) = 1              pos : neg    =     11.2 : 1.0
         count (bad) = 5              neg : pos    =     10.8 : 1.0
      count (stupid) = 2              neg : pos    =     10.8 : 1.0
      count (boring) = 2              neg : pos    =     10.4 : 1.0
      count (nature) = 2              pos : neg    =      8.5 : 1.0
   count (different) = 2              pos : neg    =      8.3 : 1.0
         count (bad) = 6              neg : pos    =      8.2 : 1.0
  count (apparently) = 2              neg : pos    =      8.0 : 1.0
        count (life) = 5              pos : neg    =      7.6 : 1.0
```

图 9-3

如果仔细检查这个列表，我们很容易发现有一些褒义词，如“wonderful（妙不可言的）”和“outstanding（杰出的）”。而单词“bad（恶劣的）”“stupid（愚蠢的）”和“boring（无趣的）”则明显是一些贬义词。如果有兴趣，读者也可以分析其他的那些特征，这将作为一个作业留给读者自己完成。以下代码摘自本书代码包中的 sentiment.py 文件。

```
import random
from nltk.corpus import movie_reviews
from nltk.corpus import stopwords
from nltk import FreqDist
from nltk import NaiveBayesClassifier
from nltk.classify import accuracy
import string

labeled_docs = [(list(movie_reviews.words(fid)), cat)
        for cat in movie_reviews.categories()
        for fid in movie_reviews.fileids(cat)]
random.seed(42)
random.shuffle(labeled_docs)

review_words = movie_reviews.words()
print("# Review Words", len(review_words))

sw = set(stopwords.words('english'))
punctuation = set(string.punctuation)

def isStopWord(word):
    return word in sw or word in punctuation

filtered = [w.lower() for w in review_words if not isStopWord(w.lower())]
print("# After filter", len(filtered))
words = FreqDist(filtered)
N = int(.05 * len(words.keys()))
```

```
word_features = words.keys()[:N]

def doc_features(doc):
    doc_words = FreqDist(w for w in doc if not isStopWord(w))
    features = {}
    for word in word_features:
        features['count (%s)' % word] = (doc_words.get(word, 0))
    return features

featuresets = [(doc_features(d), c) for (d,c) in labeled_docs]
train_set, test_set = featuresets[200:], featuresets[:200]
classifier = NaiveBayesClassifier.train(train_set)
print("Accuracy", accuracy(classifier, test_set))

print(classifier.show_most_informative_features())
```

9.8 创建词云

阅读本书前，我们也许已在 Wordle 或其他地方见过词云了；如果没见过也不要紧，因为可以在本章看个够了。虽然 Python 有两个程序库都可以用来生成词云，但是它们生成词云的效果尚不能与 Wordle 网站的效果相媲美。所以，我们不妨利用 Wordle 网站的页面来创建词云。利用 Wordle 生成词云时，需要提供一个单词列表及其对应的权值，具体格式如下。

```
Word1 : weight
Word2 : weight
```

现在，对前面例子中的代码稍作修改，让它输出这个字表。这里，我们将使用词频作为度量指标，选取词频排名靠前的单词。实际上，这里不需要任何新的代码，具体内容可以参考本书代码包中的 ch-09.ipynb 文件。

```
from nltk.corpus import movie_reviews
from nltk.corpus import stopwords
from nltk import FreqDist
import string

sw = set(stopwords.words('english'))
punctuation = set(string.punctuation)
def isStopWord(word):
    return word in sw or word in punctuation
review_words = movie_reviews.words()
```

```
filtered = [w.lower() for w in review_words if not isStopWord(w.lower())]

words = FreqDist(filtered)
N = int(.01 * len(words.keys()))
tags = words.keys()[:N]

for tag in tags:
    print(tag, ':', words[tag])
```

将上面代码的输出结果复制粘贴到上面提到的 Wordle 页面，就能得到如图 9-4 所示的词云了。

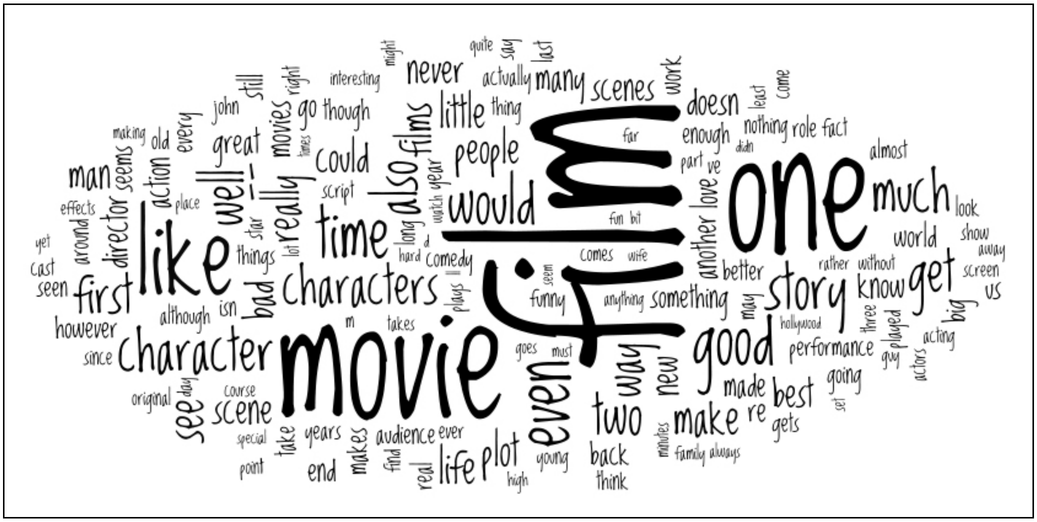


图 9-4

仔细研究这个词云，会发现它远非完美，还有很大的改进空间。因此，我们不妨做进一步的尝试。

- 进一步过滤：我们应当剔除包含数字符号和姓名的那些单词。这时，可以借助 NLTK 的 names 语料库。此外，对于在所有语料库中仅出现一次的那些单词，我们也可以置之不理，因为它们不大可能提供足够有价值的信息。
- 使用更好的度量指标：词频和逆文档频率（The Term Frequency-Inverse Document Frequency，TF-IDF）看起来是一个不错的选择。

度量指标 TF-IDF 可以通过对语料库中的单词进行排名并据此赋予这些单词相应的权重。这个权重的值与单词在特定文档中出现的次数即词频成正比。同时，它还与语料库中含有该单词的文档数量成反比，即逆文档频率。TF-IDF 的值为词频和逆文档频率之积。如

果需要自己动手实现 TF-IDF，那么还必须考虑对数标定处理。幸运的是，实际上我们根本无需考虑这些实现方面的细节，因为 scikit-learn 已经为我们准备好了一个 TfidfVectorizer 类，它有效地实现了 TF-IDF。该类能够生成一个稀疏的 SciPy 矩阵，用术语来说就是一个词条-文档矩阵，这个矩阵存放的是单词和文档的每种可能组合的 TF-IDF 值。因此，对于一个含有 2 000 个文档和 25 000 个不重复的词的语料库，可以得到一个 2 000×25 000 的矩阵。由于大部分矩阵值都为 0，我们将它作为一个稀疏矩阵来处理比较省劲。对每个单词来说，其最终的排名权重可以通过对其 TF-IDF 值求和来得到。

下面我们通过 isalpha()方法和姓名语料库来改善过滤效果，代码如下。

```
all_names = set([name.lower() for name in names.words()])

def isStopWord(word):
    return (word in sw or word in punctuation) or not word.isalpha() or
word in all_names
```

下面创建一个 NLTK 的 FreqDist 类，从而过滤掉那些只出现一次的单词。对于 TfidfVectorizer 类，需要为其提供一个字符串列表来指出语料库中的各个文档。

下面创建这个列表，代码如下。

```
for fid in movie_reviews.fileids():
    texts.append(" ".join([w.lower() for w in movie_reviews.words(fid) if
not isStopWord(w.lower()) and words[w.lower()] > 1]))
```

创建向量化程序，为了保险起见，令其忽略停用词。

```
vectorizer = TfidfVectorizer(stop_words='english')
```

创建稀疏的词条-文档矩阵。

```
matrix = vectorizer.fit_transform(texts)
```

为每个单词的 TF-IDF 值求和并将结果存放到 NumPy 数组中。

```
sums = np.array(matrix.sum(axis=0)).ravel()
```

下面通过单词的排名权值来创建一个 Pandas DataFrame 并进行相应的排序，代码如下。

```
ranks = []

for word, val in itertools.izip(vectorizer.get_feature_names(), sums):
```

```
    ranks.append((word, val))

    df = pd.DataFrame(ranks, columns=["term", "tfidf"])
    df = df.sort(['tfidf'])
    print(df.head())
```

排名最低的值将被输出，同时供将来过滤用。

```
term   tfidf
8742            greys 0.03035
2793      cannibalize 0.03035
2408          briefer 0.03035
19977 superintendent 0.03035
14022           ology 0.03035
```

接下来，我们输出排名靠前的单词，这样就可以利用 Wordle 网站生成如图 9-5 所示的词云了。



图 9-5

令人遗憾的是，我们必须亲自运行代码，才能看到上面词云中的不同色彩。相较于单调的词频而言，TF-IDF 度量指标更富于变化，因此我们就能得到更加丰富的色彩。此外，这样还能使得云雾中的单词看起来联系更加紧密。下列代码摘自本书代码包中的 ch-09.ipynb 文件。

```
from nltk.corpus import movie_reviews
from nltk.corpus import stopwords
from nltk.corpus import names
from nltk import FreqDist
```

```
from sklearn.feature_extraction.text import TfidfVectorizer
import itertools
import pandas as pd
import numpy as np
import string

sw = set(stopwords.words('english'))
punctuation = set(string.punctuation)
all_names = set([name.lower() for name in names.words()])

def isStopWord(word):
    return (word in sw or word in punctuation) or not word.isalpha() or
word in all_names

review_words = movie_reviews.words()
filtered = [w.lower() for w in review_words if not isStopWord(w.lower())]

words = FreqDist(filtered)

texts = []

for fid in movie_reviews.fileids():
    texts.append(" ".join([w.lower() for w in movie_reviews.words(fid) if
not isStopWord(w.lower()) and words[w.lower()] > 1]))

vectorizer = TfidfVectorizer(stop_words='english')
matrix = vectorizer.fit_transform(texts)
sums = np.array(matrix.sum(axis=0)).ravel()

ranks = []

for word, val in itertools.izip(vectorizer.get_feature_names(), sums):
    ranks.append((word, val))

df = pd.DataFrame(ranks, columns=["term", "tfidf"])
df = df.sort(['tfidf'])
print(df.head())

N = int(.01 * len(df))
df = df.tail(N)

for term, tfidf in itertools.izip(df["term"].values, df["tfidf"].values):
    print(term, ":", tfidf)
```

9.9 社交网络分析

所谓社交网络分析，实际上就是利用网络理论来研究社会关系，其中，网络中的节点

代表的是网络中的参与者。节点之间的连线代表的是参与者之间的相互关系。

严格来讲，这应该称为一个图。本书仅介绍如何利用流行的 Python 库 NetworkX 来分析简单的图。这里，通过 Python 库 matplotlib 来对这些网络图进行可视化。

为了安装 NetworkX，可以使用如下命令。

```
$ pip3 install networkx
```

下面导入 NetworkX 并指定一个简单的别名。

```
import networkx as nx
```

NetworkX 提供了许多示例图，下面将其列出，代码如下。

```
print([s for s in dir(nx) if s.endswith('graph')])
```

下面导入 Davis Southern women 图并绘制出各个节点的度的柱状图，代码如下。

```
G = nx.davis_southern_women_graph()
plt.figure(1)
plt.hist(nx.degree(G).values())
```

最终得到如图 9-6 所示的柱状图。

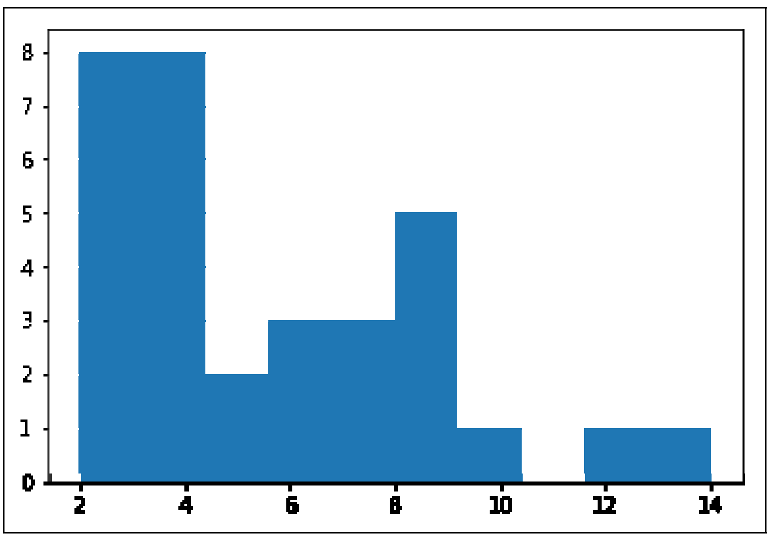


图 9-6

下面来绘制带有节点标签的网络图，所需命令如下。

```
plt.figure(2)
pos = nx.spring_layout(G)
nx.draw(G, node_size=9)
nx.draw_networkx_labels(G, pos)
plt.show()
```

得到的图形如图 9-7 所示。

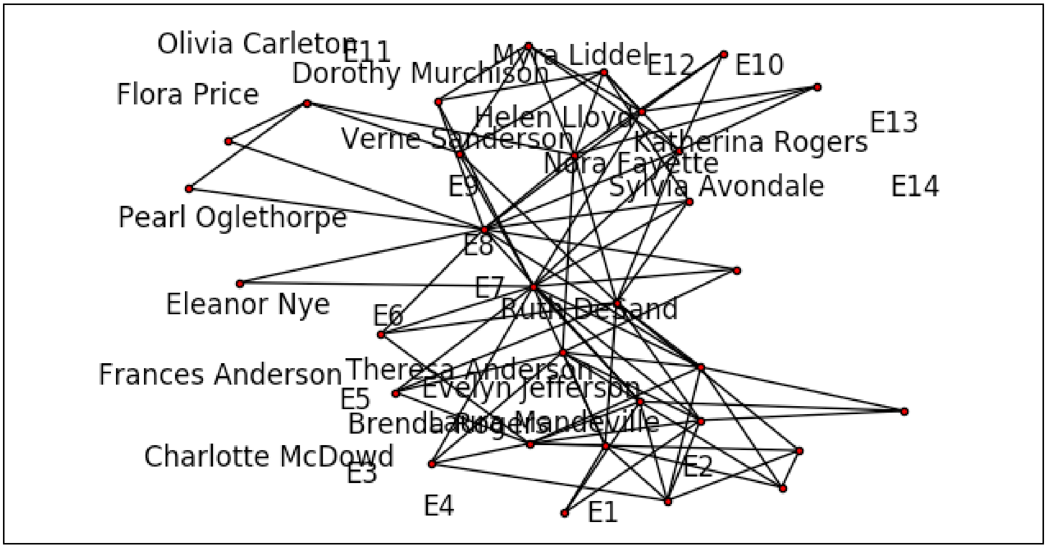


图 9-7

虽然这个例子很短，但是对于简单体验 NetworkX 的功能特性来说已经足够了。我们可以借助 NetworkX 来探索、可观化和分析社交媒体网络，如 Twitter、Facebook 以及 LinkedIn 等。当然，本节讲述的方法不仅适用于社交网络，实际上，所有类似的网络图都可以使用 NetworkX 来处理。

9.10　小结

本章讲述了文本分析方面的知识。首先，我们给出了文本分析中剔除停用词的一个最佳实践。

介绍词袋模型时，我们还创建了一个词袋来存放文档内出现的单词。此外，我们还根据所有的单词计数，为每个文档生成了一个特征向量。

分类算法是机器学习算法中的一种，用来给特定事物进行分类。朴素贝叶斯分类是一种概率算法，它基于概率与数理统计中的贝叶斯定理。这里，贝叶斯定理指出，后验概率与先验概率和相似度之积成正比。

第 10 章将对机器学习进行更深入的介绍，因为这是一个充满了无限希望的研究领域。也许有一天，它将完全替代人类劳动力。届时，以气象数据为例，来说明 Python 机器学习库 scikit-learn 的具体使用方法。

第 10 章 预测性分析与机器学习

最近，预测性分析与机器学习已经纳入许多行业的主流数据科学和数据分析的行列。相对于其他领域而言，我们对这两个领域取得突飞猛进发展的期待要更热切一些。甚至有人预言，机器学习的发展速度将日益加快，因此，几十年内，人工就会被智能机器所替代。当然，就目前人工通用智能（AGI）的发展现状来看，这只是一个遥远的乌托邦而已；但是机器学习已经取得了长足的进步，可用于自驾车、聊天机器人和 AI 助手，如亚马逊的 Alexa、苹果的 Siri 以及 Ok Google。然而，目前即使是进行非常简单的判断，如判断网络图片中是否含有猫或狗等，都需要大量的运算和数据作为支撑。预测性分析则需要借助各种各样的技术，包括机器学习，才能做出有用的判断。例如，某客户是否有能力偿还其贷款，或者某位女性客户是否有孕在身。

为了完成这些预测，需要从海量数据中提取特征。关于特征，我们之前也曾经提到过，它们又被称为预测变量。进行预测时，特征通常用于输入变量。实质上，特征可以从数据中提取，然后要做的是，找到一个函数，将特征映射到目标上。当然，这个目标可能是已知的，也可能是未知的。寻找合适的函数很难，为此，通常需要把多种不同的算法和模型组合在一起，也就是所谓的集成。集成的输出结果可以是一组模型投票决出的结果，也可以是所有结果的一个折中。但是，我们还可以使用另外一种更加高级的算法来获得最终结果。虽然我们不会在本章使用集成技术，但是大家还是有必要记住这种技术。

实际上，在前面的章节中我们已经接触过机器学习算法了，朴素贝叶斯分类算法便是其中之一。我们可以将机器学习分为下列几种类型。

- 监督学习：要求为训练数据提供标签，也就是说需要给算法提供已经分好类的样本。利用带标签的训练数据，我们可以创建一个函数，将输入变量映射为相应的输出变

量。比如，如果想对垃圾邮件进行分类，那么我们必须提供垃圾邮件和正常电子邮件的相应样本。监督学习算法的例子包括线性回归、逻辑回归、状态向量机、随机森林、K 最近邻算法等。

- 无监督学习：这类机器学习算法不需要人工输入。它能够自行发现数据中存在的模式，如大型数据集中的聚类等。无监督学习算法的例子包括 K 均值和分层聚类。
- 强化学习：这种类型的学习技术无需进行辅导，但是需要我们提供一些反馈信息。例如一台电脑可以跟它自己下棋，或者玩 1983 的《战争游戏》电影中的井字游戏和热核战争游戏。

下面我们以天气预报为例进行说明。本章会大量用到一个 Python 程序库，即 scikit-learn。虽然这个程序库提供了许多聚类分析算法、回归算法和分类算法，但是，还有一些机器学习算法未包含在 scikit-learn 中，因此，我们还需要用到其他一些 API。本章涉及的主题如下。

- 预处理
- 基于逻辑回归的分类
- 基于支持向量机的分类
- 基于 ElasticNetCV 的回归分析
- 支持向量回归
- 基于相似性传播（affinity propagation）的聚类分析
- 均值漂移算法
- 遗传算法
- 神经网络
- 决策树算法

10.1 预处理

在 scikit-learn 库中提供了一个预处理模块，下面我们将详细介绍该模块。在第 9 章中，我们已经安装了 scikit-learn 并且已经做过一次数据预处理了，即过滤掉剔除字（Stopwords）。

一些机器学习算法对某些数据比较头疼，因为这些数据不服从高斯分布，即不满足数学期望为 0、标准方差为 1 的条件。模块 sklearn.preprocessing 从而应运而生，本节将详细介绍这个模块的使用方法，我们会针对来自荷兰皇家气象学会的气象数据进行预处理。比尔特（DE BILT）气象研究中心的原始数据较多，我们所要的数据只是原始数据文件中的一列而已，这一列记录的是日降雨量。需要的数据将使用第 5 章中介绍的.npy 格式来存放。下面把数据加载到一个 NumPy 数组中。这些整数值必须全部乘以 0.1，从而得到以毫米为单位的日降水量。

该数据有一个很诡异的地方，那就是凡是小于 0.05mm 的数值，都将用作−1。因此，我们把这些数值全部设为 0.025，即 0.05 的一半。在原始数据中，如果某一天的降雨量低于这些数值，那么这一天将不会记录在案。对于这样的缺失数据，我们将完全忽略。我们可以那么做，因为本来缺失的数据点就非常多，所以也不差这一点。例如，本世纪初大约缺失一年的数据，同时后来许多天的数据也是缺失的。在 preprocessing 模块中，它的 Imputer 类提供了许多处理缺失数据的默认策略。不过，这些策略对于本节讨论的内容而言，好像不太合适。数据分析如同一扇透视数据的窗口，一扇通向知识的窗口。数据的清洗和填补可以使我们的窗口看起来更加清爽。可是，我们应该注意不要过于扭曲原始数据。

这里的机器学习样本的主要特征是一个数组，用来存放表示一年中各天（1～366）日期的数值。这些可以帮助解释季节影响因素。

期望值、标准差和安德森-达林（Anderson-Darling）检验结果（详情请参阅第 4 章）如下。

```
Rain mean 2.17919594267
Rain variance 18.803443919
Anderson rain (inf, array([ 0.576,  0.656,  0.787, 0.918,
1.092]), array([ 15. ,  10. ,   5. ,   2.5,   1. ]))
```

透过以上输出内容，可以得出可靠的结论，该数据不满足数学期望为 0 标准方差为 1 的要求，因此它不符合正态分布。在这个数据集中，0 值占比很大，说明在这些天没有下雨。大的降雨量已经越来越罕见（这倒是一件好事）。数据分布情况完全不对称，因此，这不是一个高斯分布。下面通过 scale()函数对数据进行缩放处理，代码如下。

```
scaled = preprocessing.scale(rain)
```

如今，数学期望和标准方差已经满足要求，只是数据的分布还是不对称。

```
Scaled mean 3.41301602808e-17
```

```
Scaled variance 1.0
Anderson scaled (inf, array([ 0.576, 0.656, 0.787, 0.918,
1.092]), array([ 15. ,  10. ,   5. ,  2.5, 1. ]))
```

有时，我们需要把特征值由数值型转换为布尔型。例如，进行文本分析时就经常需要进行此类变换，以简化计算。这时，可以借助 binarize()函数进行类型转换，代码如下。

```
binarized = preprocessing.binarize(rain)
print(np.unique(binarized), binarized.sum())
```

默认情况下，这将生成一个新数组；此外，我们还让它进行了某些运算。默认的阈值为 0。也就是说，正值用 1 替代，负值用 0 替代。

```
[ 0. 1.] 24594.0
```

进行分类时，类 LabelBinarizer 可以用整数来标注类别。

```
lb = preprocessing.LabelBinarizer()
lb.fit(rain.astype(int))
print(lb.classes_)
```

上述代码的输出 0～62 的一组整数。相关代码摘自本书代码包中的 ch-10.ipynb 文件。

```
import numpy as np
from sklearn import preprocessing
from scipy.stats import anderson

rain = np.load('rain.npy')
rain = .1 * rain
rain[rain < 0] = .05/2
print("Rain mean", rain.mean())
print("Rain variance", rain.var())
print("Anderson rain", anderson(rain))

scaled = preprocessing.scale(rain)
print("Scaled mean", scaled.mean())
print("Scaled variance", scaled.var())
print("Anderson scaled", anderson(scaled))

binarized = preprocessing.binarize(rain)
print(np.unique(binarized), binarized.sum())

lb = preprocessing.LabelBinarizer()
```

```
lb.fit(rain.astype(int))
print(lb.classes_)
```

10.2 基于逻辑回归的分类

逻辑回归是一种分类算法，这种算法可以用于预测事件发生的概率或者某事物属于某一类别的概率。对于多元分类问题，可以将其简化为二元分类问题。在最简单的情形下，某一类别的概率高，则意味着另一个类别的概率低。逻辑回归是以 logistic 函数为基础的，该函数的取值介于 0～1 中间，这与概率值正好吻合。因此，可以利用 logistic 函数将任意值转换为一个概率。

为了使用逻辑回归进行分类，需要定义一个相应的函数。下面开始创建分类器对象，代码如下。

```
clf = LogisticRegression(random_state=12)
```

参数 random_state 的作用，类似为随机数生成器指定的种子。之前已经讲过，交叉验证在避免过拟合方面有着非常重要的作用。k-折交叉验证是一种交叉验证技术，它会把数据集随机分为 k（一个小整数）份，每一份称为一个包。在这 k 次迭代过程中，每个包会有 1 次被用于验证，其余 9 次用于训练。对于 scikit-learn 来说，它每个类的默认 k 值都是 3，但是，通常需要将这个值设置得更大一些，如 5 或 10。迭代的结果可以在最后进行合并。对于 k-折交叉验证，scikit-learn 专门提供了一个 KFold 类。为了创建一个具有 10 个包的 KFold 对象，可以使用下列代码。

```
kf = KFold(len(y), n_folds=10)
```

然后，使用 fit()函数训练数据，代码如下。

```
clf.fit(x[train], y[train])
```

为了衡量分类的准确性，可以使用 score()方法，具体如下。

```
scores.append(clf.score(x[test], y[test]))
```

在这个例子中，可以把日期和日降雨量作为特征，下面使用这些特征来构造一个数组，代码如下。

```
x = np.vstack((dates[:-1], rain[:-1]))
```

因为要进行分类，所以首先要定义无降雨天，即降雨量为 0；然后用−1 表示有微弱降雨的小雨天；最后剩下的就是雨天。我们把这 3 种类别与数据值的符号关联起来。

```
y = np.sign(rain[1:])
```

据此，得到的平均准确率为 57%。在 scikit-learn 的数据集上，我们的平均准确率为 41%，代码具体参考本书代码包中的 log_regress.py 文件。

```
from sklearn.linear_model import LogisticRegression
from sklearn.cross_validation import KFold
from sklearn import datasets
import numpy as np

def classify(x, y):
    clf = LogisticRegression(random_state=12)
    scores = []
    kf = KFold(len(y), n_folds=10)
    for train,test in kf:
      clf.fit(x[train], y[train])
      scores.append(clf.score(x[test], y[test]))

    print("accuracy", np.mean(scores))

rain = np.load('rain.npy')
dates = np.load('doy.npy')

x = np.vstack((dates[:-1], rain[:-1]))
y = np.sign(rain[1:])
classify(x.T, y)

#iris example
iris = datasets.load_iris()
x = iris.data[:, :2]
y = iris.target
classify(x, y)
```

10.3 基于支持向量机的分类

支持向量机（Support Vector Machines，SVM）不仅可以用来进行回归分析，如支持向量回归（Support Vector Regression，SVR），而且还可以用来进行分类，支持向量机算法是 Vladimir Vapnik 于 1993 年发明出来的。

SVM 可以把数据点映射到多维空间的数据点，这种映射是通过核函数来完成的。这里的核函数既可以是线性的，也可以是非线性的。这样，分类问题就简化为寻找一个将空间一分为二的超平面，或者是能够将数据点恰如其分地划分到不同空间（类别）的多个超平面。利用超平面进行分类是一件非常困难的事情，因为这会引出软间隔的概念。软间隔用来表示对错误分类的容忍度，通常由一个用 C 表示的常量给出。另一个重要的参数就是核函数的类型，有如下几种。

- 线性函数
- 多项式函数
- 径向基函数
- Sigmoid 函数

网格搜索法可以用来寻找合适的参数，这是一种系统性的方法，即尝试所有可能的参数组合。为了进行网格搜索，可以借助 scikit-learn 提供的 GridSearchCV 类。使用这个类时，我们可以通过字典来提供分类器或回归器的类型对象。字典的键就是我们将要调整的参数，而字典的值就是需要尝试的参数值的相应列表。对于用来进行分类和回归分析的那些类，scikit-learn API 都有对应的、为其提供了添加交叉验证功能的类。不过，默认情况下，交叉验证功能都未启用。下面创建一个 GridSearchCV 对象，代码如下。

```
clf = GridSearchCV(SVC(random_state=42, max_iter=100), {'kernel':
['linear', 'poly', 'rbf'], 'C':[1, 10]})
```

上面的代码规定了最多迭代次数，这个数字不要太大，否则对我们的耐心绝对是一个严峻的考验。此外，这里没有启用交叉验证，为的是加速处理过程。而且，我们还指定了核方法的类型以及软间隔的相应参数。

上述代码为可能的参数变异创建了一个 2×3 的网格。如果时间宽裕，我们可以创建一个更大的网格来包含更多的可能值。另外，我们可以把 GridSearchCV 的 cv 参数的值设为预期的包总量，如 5 或 10。实际上，最多迭代次数设置得越高越好。至于不同的核方法，可以根据拟合的实际情况择机而用。如果把参数 verbose 设置为一个非零整数，那就可以得到每个参数值组合更为详尽的输出信息，如执行时间等。通常情况下，我们希望按照幅值大小的顺序，如 1～10 000，依次变换 soft-margin 参数的取值，这时可以借助 NumPy 的 logspace()函数来完成。

使用这个分类器时，在天气数据上得到的准确性是 56%，在鸢尾花样本数据集上得到

的准确性是 82%。GridSearchCV 的 grid_scores_区保存有网格搜索的成效数据。对于天气数据，成效数据如下。

```
[mean: 0.42879, std: 0.11308, params: {'kernel': 'linear', 'C': 1},
 mean: 0.55570, std: 0.00559, params: {'kernel': 'poly', 'C': 1},
 mean: 0.36939, std: 0.00169, params: {'kernel': 'rbf', 'C': 1},
 mean: 0.30658, std: 0.03034, params: {'kernel': 'linear', 'C':10},
 mean: 0.41673, std: 0.20214, params: {'kernel': 'poly', 'C': 10},
 mean: 0.49195, std: 0.08911, params: {'kernel': 'rbf', 'C': 10}]
```

对于鸢尾花样本数据来说，我们的成绩如下。

```
[mean: 0.80000, std: 0.03949, params: {'kernel': 'linear', 'C': 1},
 mean: 0.58667, std: 0.12603, params: {'kernel': 'poly', 'C': 1},
 mean: 0.80000, std: 0.03254, params: {'kernel': 'rbf', 'C': 1},
 mean: 0.74667, std: 0.07391, params: {'kernel': 'linear', 'C':10},
 mean: 0.56667, std: 0.13132, params: {'kernel': 'poly', 'C': 10},
 mean: 0.79333, std: 0.03467, params: {'kernel': 'rbf', 'C': 10}]
```

相关代码取自本书代码包中的 ch-10.ipynb 文件。

```
from sklearn.svm import SVC
from sklearn.grid_search import GridSearchCV
from sklearn import datasets
import numpy as np
from pprint import PrettyPrinter

def classify(x, y):
    clf = GridSearchCV(SVC(random_state=42, max_iter=100), {'kernel':
['linear', 'poly', 'rbf'], 'C':[1, 10]})

    clf.fit(x, y)
    print("Score", clf.score(x, y))
    PrettyPrinter().pprint(clf.grid_scores_)
rain = np.load('rain.npy')
dates = np.load('doy.npy')

x = np.vstack((dates[:-1], rain[:-1]))
y = np.sign(rain[1:])
classify(x.T, y)

#iris example
```

```
iris = datasets.load_iris()
x = iris.data[:, :2]
y = iris.target
classify(x, y)
```

10.4　基于 ElasticNetCV 的回归分析

弹性网络正则化（Elastic Net Regularization）是一种降低回归分析的过拟合风险的方法。弹性网络正则化实际上就是 LASSO（The Least Absolute Shrinkage and Selection Operator，LASSO）算法和岭回归方法的线性组合。LASSO 能够有效约束 L1 范数或曼哈顿距离。对于两点来说，L1 范数表示的是它们坐标值之差的绝对值之和。岭回归方法使用 L1 范数的平方作为惩罚项。处理回归问题时，拟合优度通常是由所谓的 R 平方的判定系数来判断的。令人遗憾的是，R 平方的定义并不统一。此外，这个名称还有误导之嫌，因为实际上它可以为负数。就拟合优度而言，最好的情况下其判定系数为 1。根据判定系数的定义看，其允许的取值范围还是不小的，但是我们的目标很明确，就是设法让它向 1 靠拢。

下面我们使用 10-折交叉验证并定义一个 ElasticNetCV 对象，代码如下。

```
clf = ElasticNetCV(max_iter=200, cv=10, l1_ratio = [.1, .5, .7, .9, .95,
.99, 1])
```

这里，ElasticNetCV 类使用了一个名为 l1_ratio 的参数，其取值为 0～1。如果该参数的值为 0，只使用岭回归算法；如果该参数的值为 1，只使用 LASSO 回归算法；否则，就使用混合算法。对于这个参数，可以给它指定单个数值，也可以给它提供一个数值列表。对于降雨数据，我们得分情况如下。

```
Score 0.0527838760942
```

这个得分表明，我们对数据训练不足，或者说产生欠拟合（underftting）。出现这种情况的原因有很多，如特征数量不足或者选用的模型不合适等。对于波士顿房价数据，针对现有的特征得分如下。

```
Score 0.683143903455
```

predict()方法可以针对新数据进行预测。对于预测的结果，我们可以使用散点图来可视化其效果。对于降雨数据，得到的散点图如图 10-1 所示。

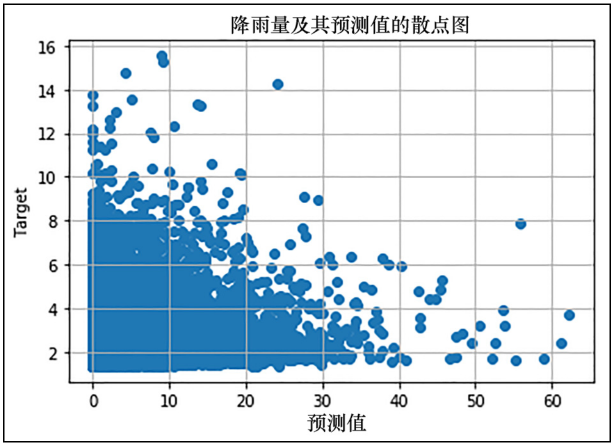


图 10-1

观察图 10-1 可以发现，这次的拟合效果不佳，即欠拟合。穿过原点的对角线，才是最理想的拟合曲线，就像图 10-2 所示的波士顿房价数据拟合情况。

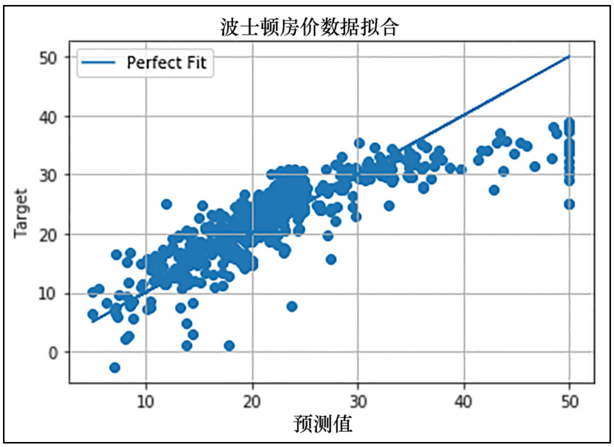


图 10-2

相关代码摘自本书代码包中的 ch-10.ipynb 文件。

```
from sklearn.linear_model import ElasticNetCV
import numpy as np
from sklearn import datasets
import matplotlib.pyplot as plt


def regress(x, y, title):
    clf = ElasticNetCV(max_iter=200, cv=10, l1_ratio = [.1, .5, .7, .9,
```

```
.95, .99, 1])

    clf.fit(x, y)
    print("Score", clf.score(x, y))

    pred = clf.predict(x)
    plt.title("Scatter plot of prediction and " + title)
    plt.xlabel("Prediction")
    plt.ylabel("Target")
    plt.scatter(y, pred)
    # Show perfect fit line
    if "Boston" in title:
        plt.plot(y, y, label="Perfect Fit")
        plt.legend()

    plt.grid(True)
    plt.show()

rain = .1 * np.load('rain.npy')
rain[rain < 0] = .05/2
dates = np.load('doy.npy')

x = np.vstack((dates[:-1], rain[:-1]))
y = rain[1:]
regress(x.T, y, "rain data")

boston = datasets.load_boston()
x = boston.data
y = boston.target
regress(x, y, "Boston house prices")
```

10.5 支持向量回归

如前所述，支持向量机也可以用于回归分析。就回归来说，我们是通过超平面而非单独的点来拟合数据的。学习曲线是一种将学习算法行为特点可视化的好办法。对于不同的训练数据量来说，学习曲线都是训练成效和测试成效所考量的一部分。创建学习曲线可能迫使我们重复训练估计器，因此会导致整体过程变慢。对此，我们可以通过使用多个并行估计器作业进行补偿。

支持向量回归是需要进行标定处理（scaling）的算法之一。

我们得到的两个高分如下。

```
Max test score Rain 0.0161004084576
Max test score Boston 0.662188537037
```

这与 ElasticNetCV 类得到的结果相仿。为此，许多 scikit-learn 类都提供了一个 n_jobs 参数。根据经验，通常系统中有几个 CPU，我们就创建几个作业。创建作业时，可以借助 Python 标准的多进程 API。为了进行训练和测试，可以调用 learning_curve()函数，代码如下。

```
train_sizes, train_scores, test_scores = learning_curve(clf, X, Y,
n_jobs=ncpus)
```

求平均数，然后画出相应的得分情况。

```
plt.plot(train_sizes, train_scores.mean(axis=1), label="Train score")
plt.plot(train_sizes, test_scores.mean(axis=1), '--', label="Test score")
```

降雨数据的学习曲线大致如图 10-3 所示。

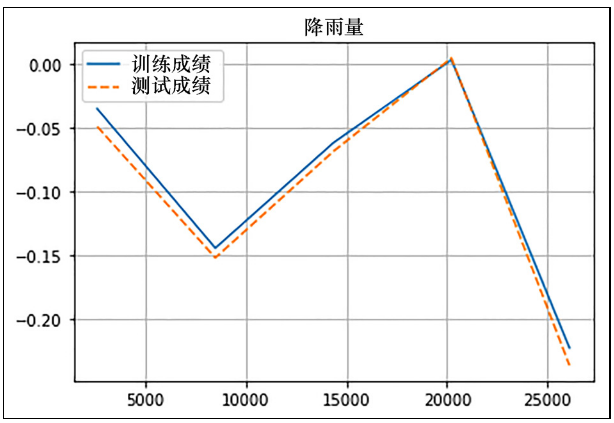


图 10-3

学习曲线在日常生活中也较常见，即练习的次数越多，学到的就越多。按照数据分析术语来讲，数据越多，学习的效果越好。如果训练成绩不错，但是测试效果不好，这就说明出现了过拟合现象。也就是说，我们的模型只适用于训练数据。图 10-4 所示是波士顿房价数据的学习曲线，它看起来更好一些。

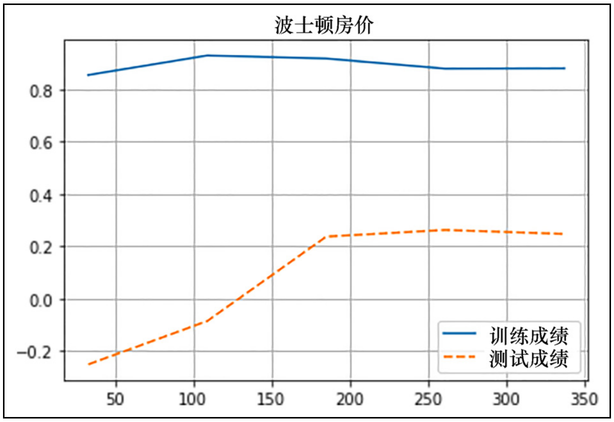


图 10-4

代码取自本书代码包中的 sv_regress.py 文件。

```
import numpy as np
from sklearn import datasets
from sklearn.learning_curve import learning_curve
from sklearn.svm import SVR
from sklearn import preprocessing
import multiprocessing
import matplotlib.pyplot as plt


def regress(x, y, ncpus, title):
    X = preprocessing.scale(x)
    Y = preprocessing.scale(y)
    clf = SVR(max_iter=ncpus * 200)

    train_sizes, train_scores, test_scores = learning_curve(clf, X, Y,
n_jobs=ncpus)

    plt.figure()
    plt.title(title)
    plt.plot(train_sizes, train_scores.mean(axis=1), label="Train score")
    plt.plot(train_sizes, test_scores.mean(axis=1), '--', label="Test
score")
    print("Max test score " + title, test_scores.max())
    plt.grid(True)
    plt.legend(loc='best')
    plt.show()
```

```
rain = .1 * np.load('rain.npy')
rain[rain < 0] = .05/2
dates = np.load('doy.npy')

x = np.vstack((dates[:-1], rain[:-1]))
y = rain[1:]
ncpus = multiprocessing.cpu_count()
regress(x.T, y, ncpus, "Rain")

boston = datasets.load_boston()
x = boston.data
y = boston.target
regress(x, y, ncpus, "Boston")
```

10.6　基于相似性传播算法的聚类分析

聚类分析，就是把数据分成一些组，这些组即所谓的聚类。聚类分析通常无需提供目标数据，从这个意义上来说，它属于无监督学习方法。一些聚类算法需要对聚类数进行推测，而另一些聚类算法则不必如此，相似性传播（Affinity Propagation，AP）算法就属于后一类算法。数据集中的各个元素都会通过特征值被映射到欧氏空间，然后计算数据点之间的欧氏距离，以此构建一个矩阵，这个矩阵就是 AP 算法的基础。因为这个矩阵可能会迅速膨胀，对此我们一定要当心，不要让它耗尽内存。实际上，scikit-learn 程序库提供了一个实用程序，来帮我们生成结构化数据。下面我们通过代码生成 3 个数据块，具体如下。

```
x, _ = datasets.make_blobs(n_samples=100, centers=3, n_features=2,
random_state=10)
```

调用 euclidean_distances()函数，创建前面提到的矩阵。

```
S = euclidean_distances(x)
```

利用这个矩阵，我们就可以给数据标注其所属的聚类了，代码如下。

```
aff_pro = cluster.AffinityPropagation().fit(S)
labels = aff_pro.labels_
```

绘制聚类后，得到的图像如图 10-5 所示。

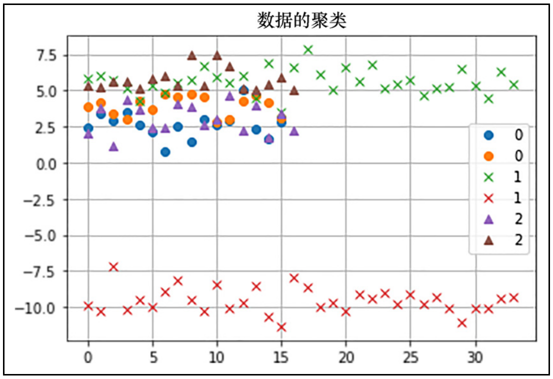


图 10-5

以上代码摘自本书代码包中的 ch-10.ipynb.py 文件。

```
from sklearn import datasets
from sklearn import cluster
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import euclidean_distances


x, _ = datasets.make_blobs(n_samples=100, centers=3, n_features=2,
random_state=10)
S = euclidean_distances(x)

aff_pro = cluster.AffinityPropagation().fit(S)
labels = aff_pro.labels_

styles = ['o', 'x', '^']

for style, label in zip(styles, np.unique(labels)):
   print(label)
   plt.plot(x[labels == label], style, label=label)
plt.title("Clustering Blobs")
plt.grid(True)
plt.legend(loc='best')
plt.show()
```

10.7 均值漂移算法

均值漂移算法是另外一种不需要估算聚类数的聚类算法。目前，这个算法已经成功应用于图像处理。该算法通过迭代来寻找一个密度函数的最大值。展示均值漂移算法前，我们首先需要使用 pandas DataFrame 计算日降雨量的平均值。下面来创建 DataFrame 并计算

其数据的平均值，代码如下。

```
df = pd.DataFrame.from_records(x.T, columns=['dates', 'rain'])
df = df.groupby('dates').mean()

df.plot()
```

结果如图 10-6 所示。

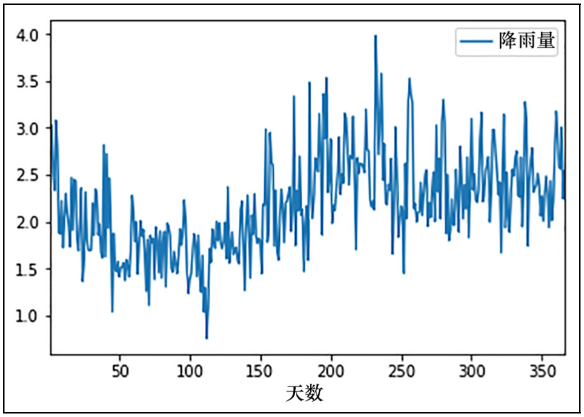


图 10-6

基于均值漂移算法的聚类代码如下所示。

```
x = np.vstack((np.arange(1, len(df) + 1) , df.as_matrix().ravel()))
x = x.T
ms = cluster.MeanShift()
ms.fit(x)
labels = ms.predict(x)
```

如果使用不同的线宽和阴影，那么绘制出的 3 个聚类如图 10-7 所示。

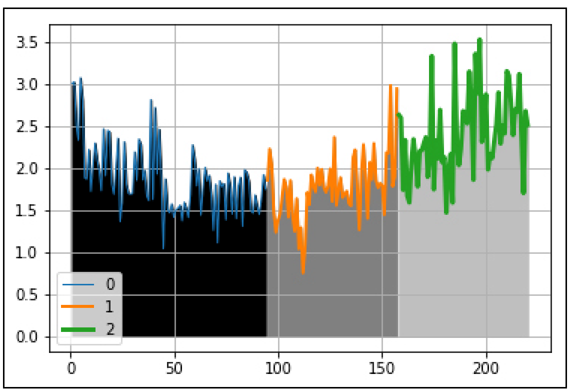


图 10-7

如你所见，我们根据年内日平均雨量（以 mm 为单位）划分出 3 个聚类。完整的代码摘自本书代码包中的 ch-10.ipynb 文件。

```
import numpy as np
from sklearn import cluster
import matplotlib.pyplot as plt
import pandas as pd

rain = .1 * np.load('rain.npy')
rain[rain < 0] = .05/2
dates = np.load('doy.npy')
x = np.vstack((dates, rain))
df = pd.DataFrame.from_records(x.T, columns=['dates', 'rain'])
df = df.groupby('dates').mean()
df.plot()
x = np.vstack((np.arange(1, len(df) + 1) , df.as_matrix().ravel()))
x = x.T
ms = cluster.MeanShift()
ms.fit(x)
labels = ms.predict(x)

plt.figure()
grays = ['0', '0.5', '0.75']

for gray, label in zip(grays, np.unique(labels)):
    match = labels == label
    x0 = x[:, 0]
    x1 = x[:, 1]
    plt.plot(x0[match], x1[match], lw=label+1, label=label)
    plt.fill_between(x0, x1, where=match, color=gray)

plt.grid(True)
plt.legend()
plt.show()
```

10.8 遗传算法

遗传算法是本书最具争议的部分，这种算法基于生物学领域的进化论。这种类型的算法在搜索和优化方面用途广泛，例如，可以使用遗传算法来搜索回归问题或分类问题的最佳参数。

在地球上，人类及其他生命形式都通过染色体来携带遗传信息。通常，我们使用字符串

对染色体进行建模。在遗传算法中，我们也沿用了这种类似的遗传信息表示方法。算法的第一步是，利用随机个体初始化种群并使用相应的表示方法表达遗传信息。此外，我们还可以使用待解决问题的已知候选解来进行初始化。之后，我们需要进入一个迭代处理过程，即所谓的逐代（generation）演化。对于每一代个体，算法都会根据预定义的适应度函数来选择一些个体，作为交配个体供下一步杂交用。适应度函数的作用是评估个体与满意解的接近程度。

下面介绍两种遗传算子（genetic operators），它们可以用来生成新的遗传信息。

- 交叉（Crossover）：这个遗传操作就是通过交配的方式来产生新个体。这里将介绍单点交叉（one-point crossover）操作，具体操作是，亲本的遗传信息相互交换，从而产生两个新的个体，第 1 个个体前半段遗传信息取自父本，后半段取自母本，第 2 个个体正好相反。例如，假设遗传信息使用 100 个列表元素来表示，那么交叉操作可以从亲本中的一方摘取前 80 个元素，然后从另一方取得后 20 个元素。当然，对于遗传算法来说，它能够通过交叉两个以上的父本和母本来繁殖新个体，这一技术正处于研究阶段（详情参见 Eiben, A. E. et al. Genetic algorithms with multi-parent recombination, Proceedings of the International Conference on Evolutionary Computation- PPSN III. The Third Conference on Parallel Problem Solving from Nature: 78-87. ISBN 3-540-58484-6, 1994）。
- 突变（Mutation）：这个遗传算子受固定突变率的限制。这个概念在好莱坞电影和大众文化中屡见不鲜。我们知道，突变虽然极为罕见，但是却能带来致命危害。不过，有时候突变也能带来我们梦寐以求的特性，而且在某些情况下，这些特性还能够遗传到下一代。

最终，新个体会取代旧个体，这时，我们就可以进入下一轮迭代了。本例中，我们将使用 Python 的 DEAP 程序库来演示遗传算法。我们安装 DEAP，具体方法如下。

```
$ pip3 install deap
```

首先，定义最大化适应度的 Fitness 子类，具体如下。

```
creator.create("FitnessMax", base.Fitness, weights=(1.0,))
```

然后，为种群中的个体定义一个模板。

```
creator.create("Individual", array.array, typecode='d',
fitness=creator.FitnessMax)
```

DEAP 中的工具箱用来注册必需的函数。下面创建一个工具箱并注册初始化函数，代码如下。

```
toolbox = base.Toolbox()
toolbox.register("attr_float", random.random)
toolbox.register("individual", tools.initRepeat, creator.Individual,
toolbox.attr_float, 200)
toolbox.register("populate", tools.initRepeat, list, toolbox.individual)
```

第 1 个函数的作用是产生浮点数，这些数的取值范围为 0～1。第 2 个函数的作用是生成一个个体，它实际上是一个由 200 个浮点数组成的列表。第 3 个函数的作用是创建由个体构成的列表，这个列表代表的是一个种群，也就是搜索或优化问题的一组可能解。

现实社会中，大部分人都很“普通”，或者说很“正常”，但是也有极少数人很“另类”，如爱因斯坦。第 3 章曾经介绍了一个 shapiro()函数，它可以用来进行正态检验。对于一个个体是否为普通个体，我们需要通过其正态检验 p 值来衡量它是普通个体的可能性有多大。下面的代码将定义一个适应度函数。

```
def eval(individual):
    return shapiro(individual)[1],
```

下面定义遗传算子，代码如下。

```
toolbox.register("evaluate", eval)
toolbox.register("mate", tools.cxTwoPoint)
toolbox.register("mutate", tools.mutFlipBit, indpb=0.1)
toolbox.register("select", tools.selTournament, tournsize=4)
```

下面对这些遗传算子进行简单解释。

- evaluate：即评估算子。这项操作用来度量各个个体的适应度。本例中，正态检验的 p 值便是适应度的衡量指标。
- mate：即交配算子。这项操作用来产生子代。本例中采用的是两点交叉。
- mutate：即突变算子。这项操作会随机修改个体。对于由布尔值构成的列表而言，这就意味着某些值会被反转，即由 True 变为 False，或者正好相反。
- mutate：即选择算子。这项操作用来选择可以进行交配的个体。

在上面的代码中，我们规定使用两点交叉，同时指定了属性被翻转的概率，下面生成一个由 400 个个体组成的原始群体，代码如下。

```
pop = toolbox.populate(n=400)
```

现在，启动进化过程，代码如下。

```
hof = tools.HallOfFame(1)
stats = tools.Statistics(key=lambda ind: ind.fitness.values)
stats.register("max", np.max)

algorithms.eaSimple(pop, toolbox, cxpb=0.5, mutpb=0.2, ngen=80,
stats=stats, halloffame=hof)
```

这个程序将提供包括每代最大适应度在内的各种统计信息。我们给它规定了交叉概率、突变率和最大代数（maximum generations），即超过这个代数后就会停止运行。下面的数据就是摘自这个程序输出的统计报告。

```
gen       nevals        max
0         400           0.000484774
1         245           0.000776807
2         248           0.00135569
...
79        250           0.99826
80        248           0.99826
```

可见，最初的分布与正态分布相去甚远，但是，最终得到的个体的分布情况如图 10-8 所示。

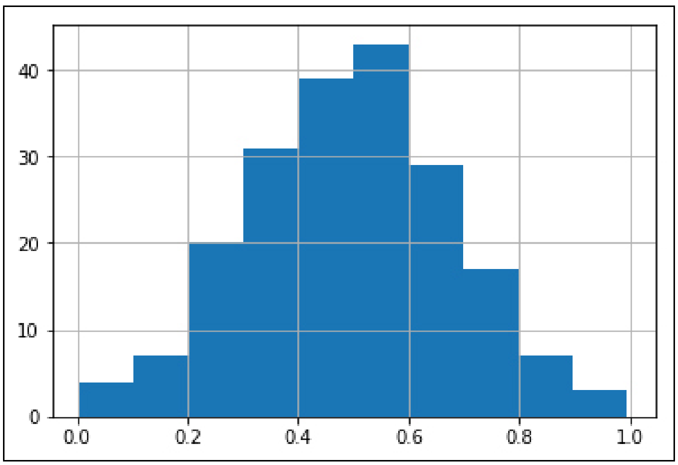


图 10-8

相关代码摘自本书代码包中的 ch-10.ipynb 文件：

```
import array
import random
import numpy as np
from deap import algorithms
```

```
from deap import base
from deap import creator
from deap import tools
from scipy.stats import shapiro
import matplotlib.pyplot as plt

creator.create("FitnessMax", base.Fitness, weights=(1.0,))
creator.create("Individual", array.array, typecode='d',
fitness=creator.FitnessMax)

toolbox = base.Toolbox()
toolbox.register("attr_float", random.random)
toolbox.register("individual", tools.initRepeat, creator.Individual,
toolbox.attr_float, 200)
toolbox.register("populate", tools.initRepeat, list, toolbox.individual)

def eval(individual):
    return shapiro(individual)[1],

toolbox.register("evaluate", eval)
toolbox.register("mate", tools.cxTwoPoint)
toolbox.register("mutate", tools.mutFlipBit, indpb=0.1)
toolbox.register("select", tools.selTournament, tournsize=4)

random.seed(42)

pop = toolbox.populate(n=400)
hof = tools.HallOfFame(1)
stats = tools.Statistics(key=lambda ind: ind.fitness.values)
stats.register("max", np.max)

algorithms.eaSimple(pop, toolbox, cxpb=0.5, mutpb=0.2, ngen=80,
stats=stats, halloffame=hof)

print(shapiro(hof[0])[1])
plt.hist(hof[0])
plt.grid(True)
plt.show()
```

10.9 神经网络

人工神经网络（ANN）的计算模型灵感来自高等动物的大脑。所谓神经网络，实际上就是由神经元组成的网络，这些神经元都具有输入和输出。例如，可以将图片像素相关的

数值作为神经网络的输入，然后将神经元的输出传递给下一个神经元，依此类推，最终得到一个多层网络。神经网络蕴涵了许多自适应元素，这使得它非常适合用于处理非线性模型和模式识别问题。接下来，我们将再次根据前些年对应某日降雨量和前一日降雨量来预测是否会下雨。因为后面要用到 Python 库 theanets，所以这里先行安装，命令如下。

```
$ pip3 install theanets nose_parameterized
```

一位技术评审在安装过程中曾经遇到问题，不过更新 NumPy 和 SciPy 后，问题就迎刃而解了。首先创建一个 Experiment，也就是创建了一个神经网络，然后再对这个网络进行训练。下面的代码将新建一个具有两个输入神经元和一个输出神经元的神经网络。

```
net = theanets.Regressor(layers=[2,3,1])
```

该网络拥有一个隐藏层，层内含有 3 个神经元；另外，这个网络还使用了 Python 标准的多进程 API 来加速运算。训练网络所需的数据包括训练样本和验证样本。

```
train = [x[:N], y[:N]]
valid = [x[N:], y[N:]]
net.train(train,valid,learning_rate=0.1,momentum=0.5)
```

如果要对验证数据进行预测，可以使用以下代码。

```
pred = net.predict(x[N:]).ravel()
```

scikit-learn 程序库提供了一个实用函数，可以用来计算分类器的准确性。下面是计算准确性的具体代码。

```
print("Pred Min", pred.min(), "Max", pred.max())
print("Y Min", y.min(), "Max", y.max())
print("Accuracy", accuracy_score(y[N:], pred >= .5))
```

注意，输出值可能每次都不一样，这是神经网络的特性所导致的。这里的输出情况如下。

```
Pred Min 0.615606596762 Max 0.615606596762
Y Min 0.0 Max 1.0
Accuracy 0.634133878385
```

相关代码摘自本书代码包中的 ch-10.ipynb 文件。

```
import numpy as np
```

```
import theanets
import multiprocessing
from sklearn import datasets
from sklearn.metrics import accuracy_score

rain = .1 * np.load('rain.npy')
rain[rain < 0] = .05/2
dates = np.load('doy.npy')
x = np.vstack((dates[:-1], np.sign(rain[:-1])))
x = x.T

y = np.vstack(np.sign(rain[1:]),)
N = int(.9 * len(x))

train = [x[:N], y[:N]]
valid = [x[N:], y[N:]]

net = theanets.Regressor(layers=[2,3,1])

net.train(train,valid,learning_rate=0.1,momentum=0.5)

pred = net.predict(x[N:]).ravel()
print("Pred Min", pred.min(), "Max", pred.max())
print("Y Min", y.min(), "Max", y.max())
print("Accuracy", accuracy_score(y[N:], pred >= .5))
```

10.10 决策树

形如 if a: else b 这样的判断语句，恐怕是 Python 程序中最常见的语句了。通过嵌套和组合这些语句，就能够建立所谓的决策树。决策树跟老式流程图非常类似，只不过流程图允许循环而已。机器学习领域中，应用决策树的过程通常被称为决策树学习。进行决策树学习时，决策树的末端节点通常又叫作叶节点，其中存放着分类问题的类标签。每个非叶节点都对应特征值之间的一个布尔条件判断。scikit-learn 使用基尼不纯度（Gini impurity）和熵作为信息的衡量指标。实际上，这两种指标衡量的是数据项被错误分类的概率。决策树非常易于理解、使用、可观化和验证。为了形象展示决策树，可以借助 Graphviz，该软件可以从 Graphviz 官网下载。此外，我们还需要安装 pydot2，命令如下。

```
$ pip3 install pydot2
```

下面把降雨数据分为训练集和测试集两部分，这需要用到 scikit-learn 提供的 train_test_split()函数，代码如下。

```
x_train, x_test, y_train, y_test = train_test_split(x, y, random_state=37)
```

创建 DecisionTreeClassifier，代码如下。

```
clf = tree.DecisionTreeClassifier(random_state=37)
```

我们可以使用 scikit-learn 的 RandomSearchCV 类来试验各种参数的取值范围，代码如下。

```
params = {"max_depth": [2, None],
              "min_samples_leaf": sp_randint(1, 5),
              "criterion": ["gini", "entropy"]}
rscv = RandomizedSearchCV(clf, params)
rscv.fit(x_train,y_train)
```

经过一番搜索后，得到的最佳成效和参数如下。

```
    Best Train Score 0.703164923517
    Test Score 0.705058763413
    Best params {'criterion': 'gini', 'max_depth': 2, 'min_samples_leaf':
2}
```

在任何情况下，即使只是想验证我们的设想，都不妨将决策树可观化，为此，可以使用下以下代码来绘制决策树的图像。

```
sio = io.StringIO()
tree.export_graphviz(rscv.best_estimator_, out_file=sio,
feature_names=['day-of-year','yest'])
dec_tree = pydot.graph_from_dot_data(sio.getvalue())

print("Best Train Score", rscv.best_score_)
print("Test Score", rscv.score(x_test, y_test))
print("Best params", rscv.best_params_)

from IPython.display import Image
Image(dec_tree.create_png())
```

最终结果如图 10-9 所示。

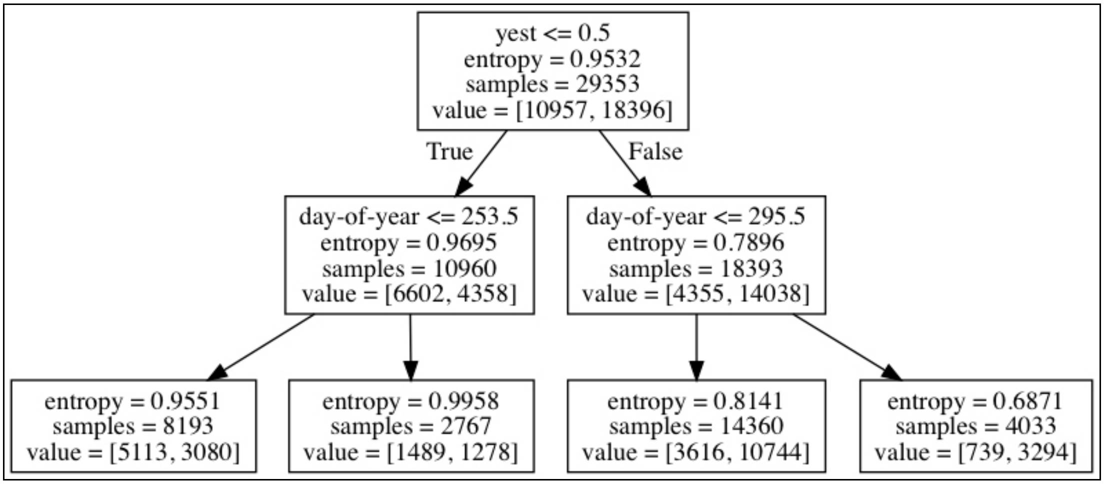


图 10-9

在上面的非叶节点中，各种判断条件都被放置在最上一行，如果条件成立，就进入左侧的子节点；否则，就进入右侧的子节点。当进入一个叶节点时，节点内最下面一行中具有最大值的类别获胜。也就是说，决策树预测输入样本属于该类别。

```
from sklearn.model_selection import train_test_split
from sklearn import tree
from sklearn.model_selection import RandomizedSearchCV
from scipy.stats import randint as sp_randint
import pydotplus as pydot
import io
import numpy as np
rain = .1 * np.load('rain.npy')
rain[rain < 0] = .05/2
dates = np.load('doy.npy').astype(int)
x = np.vstack((dates[:-1], np.sign(rain[:-1])))
x = x.T

y = np.sign(rain[1:])

x_train, x_test, y_train, y_test = train_test_split(x, y,
random_state=37)

clf = tree.DecisionTreeClassifier(random_state=37)
params = {"max_depth": [2, None],
              "min_samples_leaf": sp_randint(1, 5),
              "criterion": ["gini", "entropy"]}
rscv = RandomizedSearchCV(clf, params)
```

```
rscv.fit(x_train,y_train)

sio = io.StringIO()
tree.export_graphviz(rscv.best_estimator_, out_file=sio,
feature_names=['day-of-year','yest'])
dec_tree = pydot.graph_from_dot_data(sio.getvalue())

print("Best Train Score", rscv.best_score_)
print("Test Score", rscv.score(x_test, y_test))
print("Best params", rscv.best_params_)

from IPython.display import Image
Image(dec_tree.create_png())
```

10.11 小结

本章致力于介绍预测建模和机器学习。对于本章涉及的这些主题，读者如果有兴趣，可以进一步参考 Packt 出版的相关图书。预测性分析通常利用多种技术（其中包括机器学习），才能做出有效的预测，如判断明天是否有雨等。

SVM 能够将数据映射到多维空间，因此，我们通过它可以把分类问题简化为寻找最佳的一个或多个超平面来区隔数据点，从而达到分类目的。

弹性网络正则化则是 LASSO 方法和岭回归方法的线性组合。处理回归问题时，拟合优度通常是通过判定系数，即所谓的 R 平方来确定的。一些聚类算法需要推测聚类数，而另一些则没有这样的要求。

对于遗传算法，我们要做的第一步是根据随机个体和遗传信息的表示方法来初始化种群。对于每一代个体，都会根据预定义的适应度函数选出一些个体用于配种。机器学习领域中，运用决策树通常叫作决策树学习。

第 11 章将向大家介绍互用性和云计算。

第 11 章 Python 生态系统的外部环境和云计算

Python 生态系统外部还有许多流行的程序设计语言，如 R、C、Java 和 Fortran 等。在本章中，我们会深入研究如何让 Python 与外部环境之间交流信息。

云计算旨在将计算能力作为一种公用设施并通过互联网提供给广大用户。也就是说，用户在本地无需购置大量高性能硬件，就能方便获得强大的计算能力。相反，云计算是一种按需付费的模式。后面，我们会讨论如何将 Python 代码放到云端来使用。云计算在这个快节奏的世界中是一个日新月异的行业。目前已经有多种云计算服务可供选择，如 PythonAnywhere。本书不会专门讨论亚马逊云计算服务（Amazon Web Services，AWS），因为就像前言中提到的那样，已经有不少书籍，如 Willi Richert 和 Luis Pedro Coelho 在 Packt Publishing 出版的《Building Machine Learning Systems with Python》一书就详细地讨论过这个主题。此外，我们还需要了解站点上的数据科学工具箱（Data Science Toolbox）。它基于 Linux 的数据分析虚拟环境，该环境既可以在本地运行，也可以放到 AWS 上使用。数据科学工具箱网站上提供了许多简单明了的使用说明，可以帮助我们利用之前安装过的 Python 程序包来搭建一个工作平台。

本章将讨论如下相关主题。

- 与 MATLAB/Octave 交换信息
- 安装 rpy2
- 连接 R
- 为 Java 传递 NumPy 数组

- 集成 SWIG 和 NumPy
- 集成 Boost 和 Python
- 通过 f2py 使用 Fortran 代码
- PythonAnywhere 云

11.1 与 MATLAB/Octave 交换信息

MATLAB 及其开源替代方案 Octave 是两款非常流行的数值计算程序和程序设计语言。Octave 和 MATLAB 的语法与 Python 非常相近。事实上，我们可以通过网站来查看它们在语法方面的比对情况，例如网上就有相关信息。

> **提示**
> Octave 的下载地址为 http://www.gnu.org/software/octave/download.html。

笔者写作本书时，Octave 的最新版本是 3.8.0。我们知道，scipy.io.savemat()函数能够将一个数组保存为一个与 Octave 和 MATLAB 格式相兼容的文件。使用这个函数时，我们需要提供文件名和一个字典作为其参数，其中字典用来提供数组的名称和数组值。下面的代码摘自本书代码包中的 ch-11.ipynb 文件。

```
import statsmodels.api as sm
from scipy.io import savemat

data_loader = sm.datasets.sunspots.load_pandas()
df = data_loader.data
savemat("sunspots", {"sunspots": df.values})
```

上述代码会将太阳黑子数据保存到名为 sunspots.mat 的一个文件中。需要注意的是，这里的文件扩展名是自动添加上的。然后，我们启动 Octave 的图形用户界面或者命令行接口并加载刚才创建的文件，将会看到如下数据。

```
octave:1> load sunspots.mat
octave:2> sunspots
sunspots =
   1.7000e+03   5.0000e+00
   1.7010e+03   1.1000e+01
```

```
1.7020e+03   1.6000e+01
```

11.2 安装 rpy2

R 语言在统计学中非常流行，它是由 C 和 Fortran 语言编写而成的，而且遵循 GNU 通用公共许可证。R 语言能够为建模、统计检验、时间序列分析、分类、可视化和聚类分析提供强有力的支持。R 语言综合典藏网（The Comprehensive R Archive Network，CRAN）及其他资料档案库网站为我们提供了数以千计的 R 程序包，来帮助我们完成各种任务。

> **提示**
> 我们可以从官网下载 R。

rpy2 程序包能够简化 Python 和 R 之间的交互操作。通过 pip 安装 rpy2 的具体命令如下。

```
$ pip3 install rpy2
```

> **提示**
> 如果之前已经安装了 rpy2，那么我们可以严格按照官网上面的技术指导来升级 rpy2，因为这个过程可能稍微有点麻烦。

11.3 连接 R

R 提供了一个 datasets 包，其中含有许多样本数据集。这当中，morley 数据集提供了许多 1879 年采集的光速测量数据。光速是一个基本的物理常数，当前已知的数值已经非常精确了。另外，光速方面的数据还可以在 scipy.constants 模块中找到。这些 R 数据存放于一个含有 3 列的 R 数据框内。

- 实验编号，范围为 1～5。

- 轮数，每次实验进行 20 轮，所以共进行 100 次测量。
- 减去 299 000 后的实测光速，这里以 km/s 为单位。

利用 rpy2.robjects.r()函数，我们就可以在 Python 环境中执行 R 代码了。

现在我们来加载数据，代码如下。

```
pandas2ri.activate()
r.data('morley')
```

此外，Pandas 库中的 pandas.rpy.common 模块也为我们提供了相应的 R 接口，因此我们建议读者使用 rpy2 的相应模块模块。

下面我们将数据加载至 pandas DataFrame，方法如下。

```
df = r['morley']
```

下列代码将根据实验对数据进行分组并生成一个 5×2 的 NumPy 数组。

```
samples = dict(list(df.groupby('Expt')))
samples = np.array([samples[i]['Speed'].values for i in samples.keys()])
```

有了源自不同实验的数据后，现在我们想看看这些实验中的数据点是否服从相同的分布。Kruskal-Wallis 单向方差分析是一种统计学方法，可以直接研究样本而无需假设其概率分布。对于这个检验来说，零假设意味着所有样本具有相同的中位数。这种检验可以通过 scipy.stats.kruskal()函数来进行。完成该检验的代码如下。

```
print("Kruskal", kruskal(samples[0], samples[1], samples[2], samples[3],
samples[4]))
```

这个检验的统计数字和 p 值如下。

```
Kruskal KruskalResult(statistic=15.022124661246552,
pvalue=0.0046555484175328015)
```

我们可以拒绝零假设，但是这样就无法得知哪一个实验或哪一些实验具有偏离的中位数（deviating median），进一步的分析将作为一项练习留给读者自己完成。如果绘制各实验的最小值、最大值和平均值，就会得到如图 11-1 所示的图形。

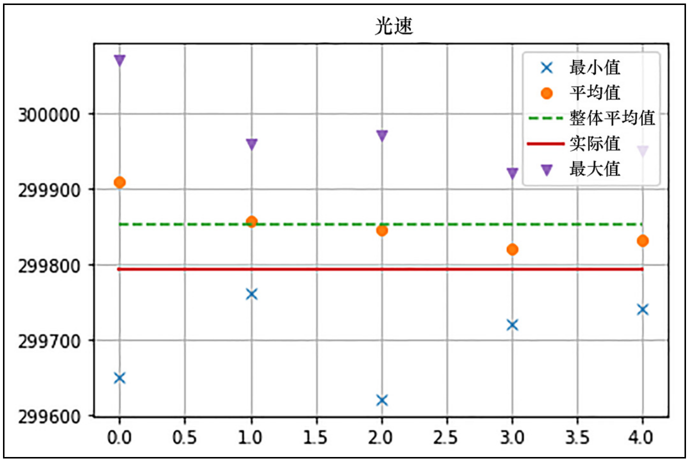


图 11-1

下列代码摘自本书代码包中的 ch-11.ipynb 文件。

```
import rpy2.robjects as ro
from rpy2.robjects import pandas2ri
from rpy2.robjects import r

from scipy.stats import kruskal
import matplotlib.pyplot as plt
import numpy as np
from scipy.constants import c

pandas2ri.activate()
r.data('morley')

df = r['morley']

df['Speed'] = df['Speed'] + 299000

samples = dict(list(df.groupby('Expt')))
samples = np.array([samples[i]['Speed'].values for i in
samples.keys()])
print("Kruskal", kruskal(samples[0], samples[1], samples[2], samples[3],
samples[4]))

plt.title('Speed of light')
plt.plot(samples.min(axis=1), 'x', label='min')
plt.plot(samples.mean(axis=1), 'o', label='mean')
plt.plot(np.ones(5) * samples.mean(), '--', label='All mean')
```

```
plt.plot(np.ones(5) * c/1000, lw=2, label='Actual')
plt.plot(samples.max(axis=1), 'v', label='max')
plt.grid(True)
plt.legend()
plt.show()
```

11.4　为 Java 传递 NumPy 数组

就像 Python 那样，Java 也是一种非常受欢迎的程序语言。第 8 章中已经安装过 Java，因为这是使用 Cassandra 的前提条件。为了运行 Java 代码，需要 Java 运行环境（Java Runtime Environment，JRE）的支持。为了开发程序，还需要用到 Java 开发工具包（Java Development Kit，JDK）。

Jython 是 Python 的纯 Java 实现。Jython 程序代码可以使用所有的 Java 类。然而，利用 C 语言编写的 Python 模块却无法导入 Jython。

这的确是一个让人头疼的问题，因为许多用于数值运算和数据分析的 Python 程序库都含有 C 语言实现的模块。不过，JPype 程序包已经为此提供了一个解决方案，要安装 JPype，请运行下列命令。

```
$ pip3 install JPype1
```

然后，通过下列命令启动 Java 虚拟机（Java Virtual Machine，JVM ）。

```
jpype.startJVM(jpype.getDefaultJVMPath())
```

下面利用随机数来创建一个名为 JArray 的 JPype 数组，代码如下。

```
values = np.random.randn(7)
java_array = jpype.JArray(jpype.JDouble, 1)(values.tolist())
```

然后，输出各个数组元素，方法如下。

```
for item in java_array:
   jpype.java.lang.System.out.println(item)
```

最后，通过下列命令来关闭 JVM。

```
jpype.shutdownJVM()
```

下列代码摘自本书代码包中的 ch-11.ipynb 文件。

```
import jpype
import numpy as np
from numpy import random
jpype.startJVM(jpype.getDefaultJVMPath())

random.seed(44)
values = np.random.randn(7)
java_array = jpype.JArray(jpype.JDouble, 1)(values.tolist())

for item in java_array:
   jpype.java.lang.System.out.println(item)

jpype.shutdownJVM()
```

在 Jupyter Notebook 中执行上述代码时，Jupyter 控制台中会输出以下内容。

```
[W 18:23:45.918 NotebookApp] 404 GET
/nbextensions/widgets/notebook/js/extension.js?v=20170305114358 (::1)
4.30ms referer=http://localhost:8888/notebooks/ch-12.ipynb
    -0.7506147172558728
    1.3163573247118194
    1.2461400286434303
    -1.6049157412585944
    -1.468143678979905
    -1.7150704579733684
    1.8587836915125544
    JVM activity report     :
    classes loaded        : 32
    JVM has been shutdown
```

11.5 集成 SWIG 和 NumPy

C 语言是从 20 世纪 70 年代发展起来的一种广泛流传的程序语言。此外，还有许许多多非标准的 C 语言，使得 C 对其他程序设计语言影响深远。C 并非面向对象的编程语言，从而 C++应运而生。C++是一种面向对象的编程语言，它继承了 C 语言的特性，因此，可以把 C++看成是 C 语言的一个超集。无论是 C 语言，还是 C++语言，都是编译型编程语言。我们需要先把源代码编译成目标文件，然后链接目标文件，得到动态共享库。幸运的是，当集成 C 和 Python 时，我们有许多的选择余地，例如，可以使用简单封装和接口生成器（Simplified Wrapper and Interface Generator，SWIG）来生成集成代码。

SWIG 在开发过程中新增了一个步骤，那就是需要生成介于 Python 和 C（或者 C++）

之间的黏合代码（Glue Code）。笔者写作本书时，SWIG 的最新版本是 3.0.12。为了安装 SWIG，还需要另外安装 PCRE（Perl Compatible Regular Expressions，PCRE）。PCRE 是一个 C 语言正则表达式程序库。笔者写作本书时，PCRE 的最新版本是 8.39。在解压下载的 PCRE 之后，可以通过下列命令进行安装。

```
$ ./configure
$ make
$ make install
```

需要注意的是，执行上面最后一条命令要求具有 root 或者 sudo 用户的访问权限。此外，这些命令同样适用于 SWIG 的安装。下面开始编写一个存放函数定义的头文件，其中定义了如下函数。

```
double sum_rain(int* rain, int len);
```

后面，我们将利用以上函数对第 10 章中分析的降水量 rain 进行求和。相关的代码摘自本书代码包中的 sum_rain.h 文件。这个函数的实现代码取自本书代码包中的 sum_rain.cpp 文件。

```
double sum_rain(int* rain, int len) {

  double sum = 0.;

  for (int i = 0; i < len; i++){
    if(rain[i] == -1) {
       sum += 0.025;
    } else {
      sum += 0.1 * rain[i];
    }
  }

  return sum;
}
```

定义如下 SWIG 接口文件（具体参见本书代码包中的 sum_rain.i 文件）。

```
%module sum_rain

%{
  #define SWIG_FILE_WITH_INIT
  #include "sum_rain.h"
%}
```

```
%include "/tmp/numpy.i"

%init %{
  import_array();
%}

%apply (int* IN_ARRAY1, int DIM1) {(int* rain, int len)};

%include "sum_rain.h"
```

上述代码需要用到接口文件 numpy.i，虽然在这个例子中，接口文件 numpy.i 是放置在 /tmp 目录下面，但是，我们实际上可以随意放置。现在，我们来生成 SWIG 黏合代码，具体如下。

```
$ swig -c++ -python sum_rain.i
```

上面的步骤将会生成一个名为 sum_rain_wrap.cxx 的文件。接下来，编译 sum_rain.cpp 文件，代码如下。

```
$ g++ -O2 -fPIC -c sum_rain.cpp -I<Python headers dir>
```

在上面的命令中，我们需要给出 Python 的 C 语言头文件所在目录。实际上，可以通过以下命令来显示该目录所在位置。

```
$ python3-config --includes
```

因此，实际上是通过下列命令进行编译的。

```
$ g++ -O2 -fPIC -c sum_rain.cpp $(python3-config --includes)
```

这个目录的位置可能会随着 Python 版本和操作系统的不同而有所变化，一般位于 /usr/include/ python3.6。编译生成的 SWIG 封装文件如下。

```
$ g++ -O2 -fPIC -c sum_rain_wrap.cxx $(python3-config --includes) -
I<numpy-dir>/core/include/
```

上面的命令依赖于 NumPy 的具体安装位置，安装位置可以通过 Python 的 Shell 命令界面找到，代码如下。

```
$ python3
>>> import numpy as np
```

```
>>> np.__file__
```

在屏幕中输出的字符串通常由 Python 版本、site-packages 和最后面的__init__.pyc 组成。实际上，我们只要除去最后的一部分，即__init__.pyc，就是 NumPy 的所在目录。另外，我们也可以通过下面的方法来取得 NumPy 目录所在位置。

```
>>> from imp import find_module
>>> find_module('numpy')
```

在电脑上输入以下命令。

```
$ g++ -O2 -fPIC -c sum_rain_wrap.cxx $(python3-config --includes) -
I/usr/local/lib/python3.6/site-packages/numpy/core/include/
```

最后，我们需要将编译生成的目标文件链接起来，具体如下。

```
$ g++ -lpython3.6 -dynamiclib sum_rain.o sum_rain_wrap.o -o _sum_rain.so -L
/usr/local/Cellar/python3/3.6.0/Frameworks/Python.framework/Versions/3.6/li
b
```

对于不同的操作系统，如 Windows 系统，上面的各个步骤可能会有所差异，除非我们使用 Cygwin。如果需要，建议读者通过 SWIG 用户邮件列表（地址为 http://www.swig.org/mail.html）或者 StackOverfow 来请求帮助。

我们可以通过本书代码包中的 swig_demo.py 文件来测试以上创建的程序。

```
from _sum_rain import *
import numpy as np

rain = np.load('rain.npy')
print("Swig", sum_rain(rain))
rain = .1 * rain
rain[rain < 0] = .025
print("Numpy", rain.sum())
```

使用以下命令执行这个文件。

```
$ python3 swig_demo.py
```

如果代码和 Python 都没有问题，将看到如下内容。

```
Swig 85291.554999999328
Numpy 85291.55
```

11.6　集成 Boost 和 Python

Boost 是一个提供了 Python 编程接口的 C++程序库，下载地址为 http://www.boost org/sers/download/。笔者写作本书时，Boost 的最新版本是 1.63.0。下面介绍一种最简单同时也是最慢的安装方法，命令如下。

```
$ ./bootstrap.sh --prefix=/path/to/boost
$ ./b2 install
```

这里，参数 prefix 用来规定安装目录。本例中，假定 Boost 已经安装到用户 home 目录下面的 Boost 子目录中，即~/Boost。在这个目录中，我们需要创建两个子目录 lib 和 include。对于 UNIX 和 Linux 系统来说，可以通过以下命令达到这个目的。

```
export LD_LIBRARY_PATH=$HOME/Boost/lib:${LD_LIBRARY_PATH}
```

对于 Mac OS X 操作系统来说，需要设置如下环境变量。

```
export DYLD_LIBRARY_PATH=$HOME/Boost/lib
```

在我们的例子中，我们将这个变量设置为以下格式。

```
export DYLD_LIBRARY_PATH=/usr/local/Cellar/boost/1.63.0/lib
```

这里，我们将重新定义一个完成降雨量求和的函数。下列代码摘自本书代码包中的 boost_rain.cpp 文件。

```
#include <boost/python.hpp>

double sum_rain(boost::python::list rain, int len) {

  double sum = 0.;

  for (int i = 0; i < len; i++){
    int val = boost::python::extract<int>(rain[i]);
    if(val == -1) {
       sum += 0.025;
    } else {
      sum += 0.1 * val;
    }
  }
```

```
  return sum;
}

BOOST_PYTHON_MODULE(librain) {
    using namespace boost::python;
    def("sum_rain", sum_rain);
}
```

上面这个函数需要两个参数，即一个 Python 列表和该列表的大小。下面演示如何调用该函数，完整代码请参考本书代码包中的 rain_demo.py 文件。

```
import numpy as np
import librain

rain_data = np.load('rain.npy')
print("Boost", librain.sum_rain(rain_data.astype(int).tolist(),
len(rain_data)))
rain_data = .1 * rain_data
rain_data[rain_data < 0] = .025
print("Numpy", rain_data.sum())
```

利用本书代码包中的 Makefile 文件，可以实现这个开发过程的自动化。

```
CC = g++
PYLIBPATH = $(shell python3-config --exec-prefix)/lib
LIB = -L$(PYLIBPATH) $(shell python3-config --libs) -
L/usr/local/Cellar/boost/1.63.0/lib -L/usr/local/Cellar/boostpython/
1.63.0/lib -lboost_python3
OPTS = $(shell python3-config --include) -O2 -
I/usr/local/Cellar/boost/1.63.0/include

default: librain.so
  @python3 ./rain_demo.py

librain.so: rain.o
  $(CC) $(LIB) -Wl,-rpath,$(PYLIBPATH) -shared $< -o $@

rain.o: boost_rain.cpp Makefile
  $(CC) $(OPTS) -c $< -o $@

clean:
  rm -rf *.so *.o

.PHONY: default clean
```

在命令行中运行以下命令。

```
$ make clean;make
```

结果如下。

```
Boost 85291.54999999328
Numpy 85291.55
```

11.7 通过 f2py 使用 Fortran 代码

Fortran 源自于“公式翻译”（Formula Translation System）的缩写，是一种成熟的编程语言，通常用于科学计算。它发源于 20 世纪 50 年代，随后的新版本包括 Fortran 77、Fortran 90、Fortran 95、Fortran 2003 以及 Fortran 2008，感兴趣的读者可以参考 http://en.wikipedia.org/wiki/Fortran 页面，以了解更多详情。它的每一个新版本都添加了新的特性和新的编程范式。对于下面的例子来说，我们需要安装一个 Fortran 编译程序。编译程序 gfortran 是一个 GNU Fortran 编译程序，读者可以自行下载。

同时，NumPy 的 f2py 模块可以为我们充当 Fortran 与 Python 之间的编程接口。如果已经安装好了 Fortran 编译程序，那么就可以借助这个模块，从 Fortran 程序代码来创建共享程序库了。这里，我们将编写一个 Fortran 子例程，来计算降雨量之和，这与前面的例子类似。好了，现在先来定义子例程并把它存放到一个 Python 字符串中。然后，调用 f2py.compile()函数，从 Fortran 代码生成一个共享程序库。完整代码请参考本书代码包中的 fort_src.py 文件。

```
from numpy import f2py
fsource = '''
       subroutine sumarray(A, N)
       REAL, DIMENSION(N) :: A
       INTEGER :: N
       RES = 0.1 * SUM(A, MASK = A .GT. 0)
       RES2 = -0.025 * SUM(A, MASK = A .LT. 0)
       print*, RES + RES2
       end
'''
f2py.compile(fsource,modulename='fort_sum',verbose=0)
```

使用以下命令执行文件来生成该模块。

```
$ python3 fort_src.py
```

调用这个子例程的完整代码请参考本书代码包中的 fort_demo.py 文件。

```
import fort_sum
import numpy as np
rain = np.load('rain.npy')
fort_sum.sumarray(rain, len(rain))
rain = .1 * rain
rain[rain < 0] = .025
print("Numpy", rain.sum())
```

使用以下命令执行文件将得到以下输出。

```
$ python3 fort_demo.py
```

下面是 Fortran 代码和 NumPy 代码生成的结果，如果忽略 Fortran 子例程计算结果的最后两位小数，结果是一致的。

```
85291.5547
Numpy 85291.55
```

11.8 PythonAnywhere 云

PythonAnywhere 是一个用于 Python 开发的云服务。PythonAnywhere 的接口是完全基于 Web 的，同时还能够模拟 Bash、Python 和 IPython 控制台界面。PythonAnywhere 环境中已经安装了许多 Python 程序库。

当然，这些版本与目前最新的稳定版本相比，多少还有些滞后。笔者写作本书时，从 PythonAnywhere 的 Bash 控制台安装 Python 软件可能会遇到一些问题，所以我们不推荐这种安装方式。当您首次访问 PythonAnywhere 的网站时，将看到的屏幕如图 11-2 所示，从这里就可以登录到 PythonAnywhere 环境。

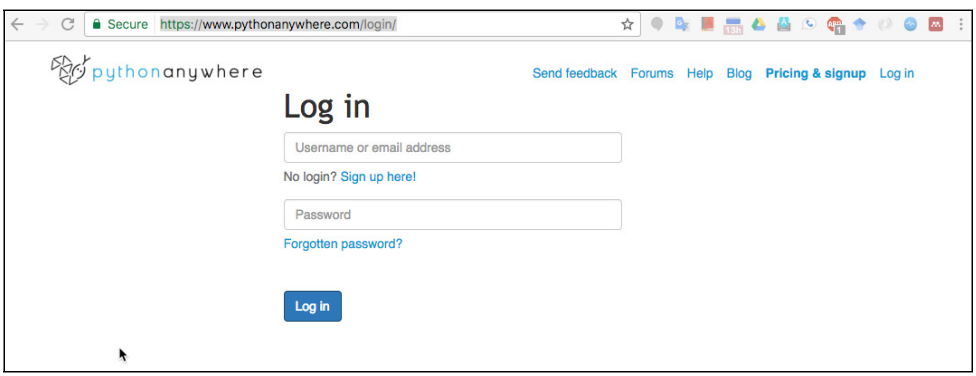


图 11-2

提交登录名和密码后，将看到如图 11-3 所示的 Web 应用程序。

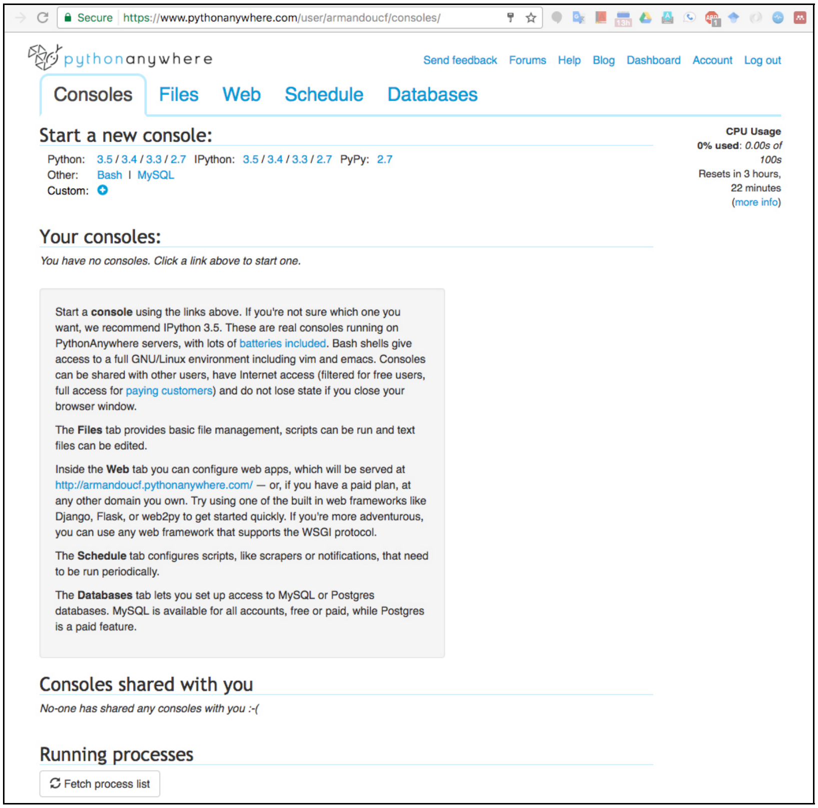


图 11-3

我们建议读者直接上传 Python 源文件，而不要通过 PythonAnywhere 工作平台安装，这是因为它不如本地程序反应得快。我们点击 Web 应用上的“Files”选项来上传文件，如图 11-4 所示。请上传 bn_demo.py 文件。

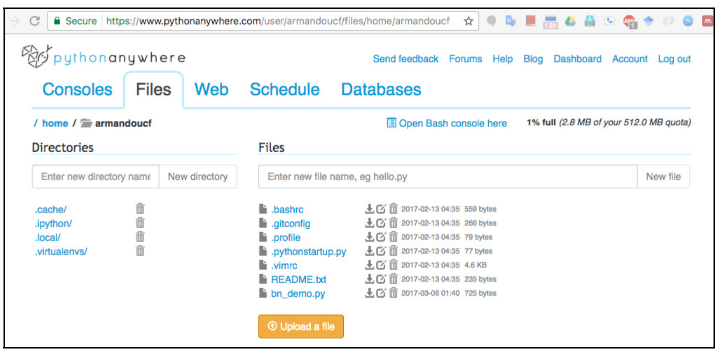


图 11-4

我们执行该程序，单击 bn_demp.py 文件，然后单击如图 11-5 所示屏幕右上角的“Run”按钮。

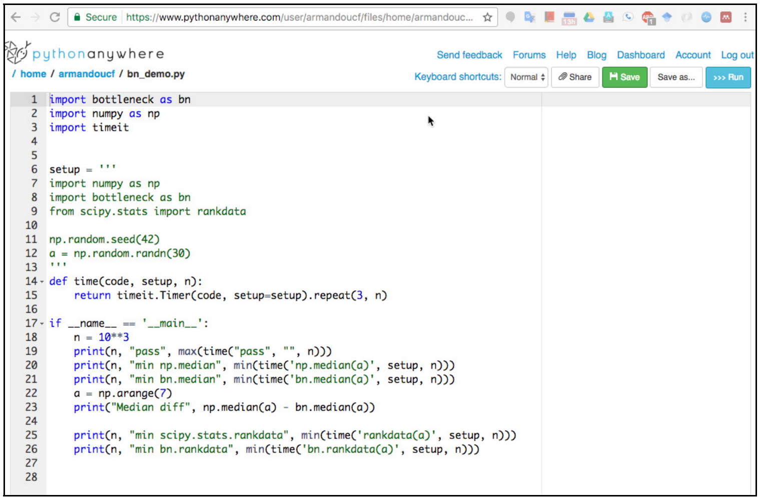


图 11-5

我们将在控制台中看到如图 11-6 所示的输出。



图 11-6

11.9　小结

在本章中，我们考察了 Python 的外部环境。在 Python 生态系统外部，还有许多非常流行的程序设计语言，如 R、C、Java 和 Fortran 等。这里，我们考察了许多个程序库，连接 Python 和其他语言时，它们都可以用来起到胶合作用，如用来连接 R 语言代码的 rpy2 库，连接 C 语言代码的 Boost 和 SWIG 库，连接 Java 语言代码的 JPype 库以及用来连接 Fortran 语言代码的 f2py 库。云计算的目的主要是将计算能力作为一种公用设施并通过互联网提供给大众。此外，本章还简要概述了当前针对 Python 的各种云计算服务，其中包括 PythonAnywhere，它是一种专门为 Python 提供的云计算服务。

第 12 章我们将把注意力放在性能的提高方面。一般来说，借助并行化或者利用 C 语言重写部分 Python 代码等代码优化手段，可以极大地提高 Python 代码的运行速度。此外，我们还会讨论各种性能分析（Profling）工具和并发性应用程序接口。

第 12 章 性能优化、性能分析与并发性

“过早优化是祸根。”

——Donald Knuth，著名的计算机科学家和数学家

在实际的应用中，性能与功能特性、健壮性、可维护性、可测性以及可用性等是一样重要的。此外，性能与应用程序的可扩展性成正比。因此，性能是本书中一个不可或缺的一个话题。这也是我们把性能这个主题放到最后来讲的原因之一。在这里，我们将性能分析（Profling）作为关键技术，介绍如何改善软件的性能。对于分布式多核系统，我们也会介绍相应的性能优化框架。

本章将讨论以下主题。

- 代码的性能分析（Profling）
- 安装 Cython
- 调用 C 代码
- 利用 multiprocessing 创建进程池
- 通过 Joblib 提高 for 循环的并发性
- 比较 Bottleneck 函数与 NumPy 函数
- 通过 Jug 实现 MapReduce
- 安装 MPI for Python
- IPython Parallel

12.1 代码的性能分析

所谓性能分析，就是以收集程序运行时的信息为手段，找出代码中哪些部分较慢或者占用内存或处理器资源较多，以便进一步对这些代码做出相应的调整。这里，我们将以第 9 章中的情绪分析代码为蓝本，稍作修改后，以此为例进行性能分析。对于这个程序，我们将按照多进程编程准则对其进行重构。本章后面的部分将对多进程技术展开进一步讨论。

此外，我们会对停用词的过滤做进一步简化。最后，我们还会在不降低准确性的前提下，来进一步减少作为特征的单词。对于最后这一点，效果最显著。原始代码的运行时间大约需要 20s。而新的代码，其速度将会有明显提升，同时我们将它作为本章的比较基准。某些代码修改与性能分析有关，这将在本节后面讲解。下列代码摘自本书代码包中的 prof_demo.py 文件。

```
import random
from nltk.corpus import movie_reviews
from nltk.corpus import stopwords
from nltk import FreqDist
from nltk import NaiveBayesClassifier
from nltk.classify import accuracy

import builtins

# Define profile function so the profile decorator
#   does not return error when code is not profiled
try:
    profile = builtins.profile
except AttributeError:
    def profile(func):
        return func

@profile
def label_docs():
    docs = [(list(movie_reviews.words(fid)), cat)
            for cat in movie_reviews.categories()
            for fid in movie_reviews.fileids(cat)]
    random.seed(42)
    random.shuffle(docs)

    return docs
```

```
@profile
def isStopWord(word):
    return word in sw or len(word) == 1

@profile
def filter_corpus():
    review_words = movie_reviews.words()
    print("# Review Words", len(review_words))
    res = [w.lower() for w in review_words if not isStopWord(w.lower())]
    print("# After filter", len(res))

    return res

@profile
def select_word_features(corpus):
    words = FreqDist(corpus)
    N = int(.02 * len(words.keys()))
    return list(words.keys())[:N]

@profile
def doc_features(doc):
    doc_words = FreqDist(w for w in doc if not isStopWord(w))
    features = {}
    for word in word_features:
        features['count (%s)' % word] = (doc_words.get(word, 0))
    return features

@profile
def make_features(docs):
    return [(doc_features(d), c) for (d,c) in docs]

@profile
def split_data(sets):
    return sets[200:], sets[:200]

if __name__ == "__main__":
    labeled_docs = label_docs()

    sw = set(stopwords.words('english'))
    filtered = filter_corpus()
    word_features = select_word_features(filtered)
    featuresets = make_features(labeled_docs)
    train_set, test_set = split_data(featuresets)
    classifier = NaiveBayesClassifier.train(train_set)
    print("Accuracy", accuracy(classifier, test_set))
```

```
print(classifier.show_most_informative_features())
print(classifier.show_most_informative_features())
```

测量运行时间时，运行的进程越少越好。但是，我们不能保证后台没有进程运行，所以需要通过 time 命令测量 3 次运行时间并以最短的时间为准。各种操作系统和 Cygwin 下面都提供了 time 命令，使用方法如下。

```
$ time python3 prof_demo.py
```

这样，我们会得到一个 real 类型的运行时间，这种测量方法采用的是时钟时间；对于 user 和 sys 类型的运行时间，则是通过 CPU 时间测量的程序运行时间。实际上，sys 时间就是在内核中耗费的时间。笔者的电脑上测得的运行时间见表 12-1，其中最小值用括号标示了出来。

表 12-1

时间类型	第一轮	第二轮	第三轮
real	11.521	10.808	(10.416)
user	9.758	9.826	(9.444)
sys	0.965	0.643	(0.620)

下面利用 Python 内置的分析工具来分析代码，具体如下。

```
$ python3 -m cProfile -o /tmp/stat.prof prof_demo.py
```

这里，-o 选项用来指定输出文件。此外，利用 PyPi 的程序包 gprof2，可以实现分析工具输出结果的可视化。下面我们安装 gprof2dot，方法如下。

```
$ pip3 install gprof2dot
```

现在创建一个 PNG 格式的可视化图形，命令如下。

```
$ gprof2dot -f pstats /tmp/stat.prof |dot -Tpng -o /tmp/cprof.png
```

> **提示**
> 如果出现错误信息 dot: command not found，表明我们尚未安装 Graphviz。

完整的图像很大，图 12-1 只展示其中一部分。

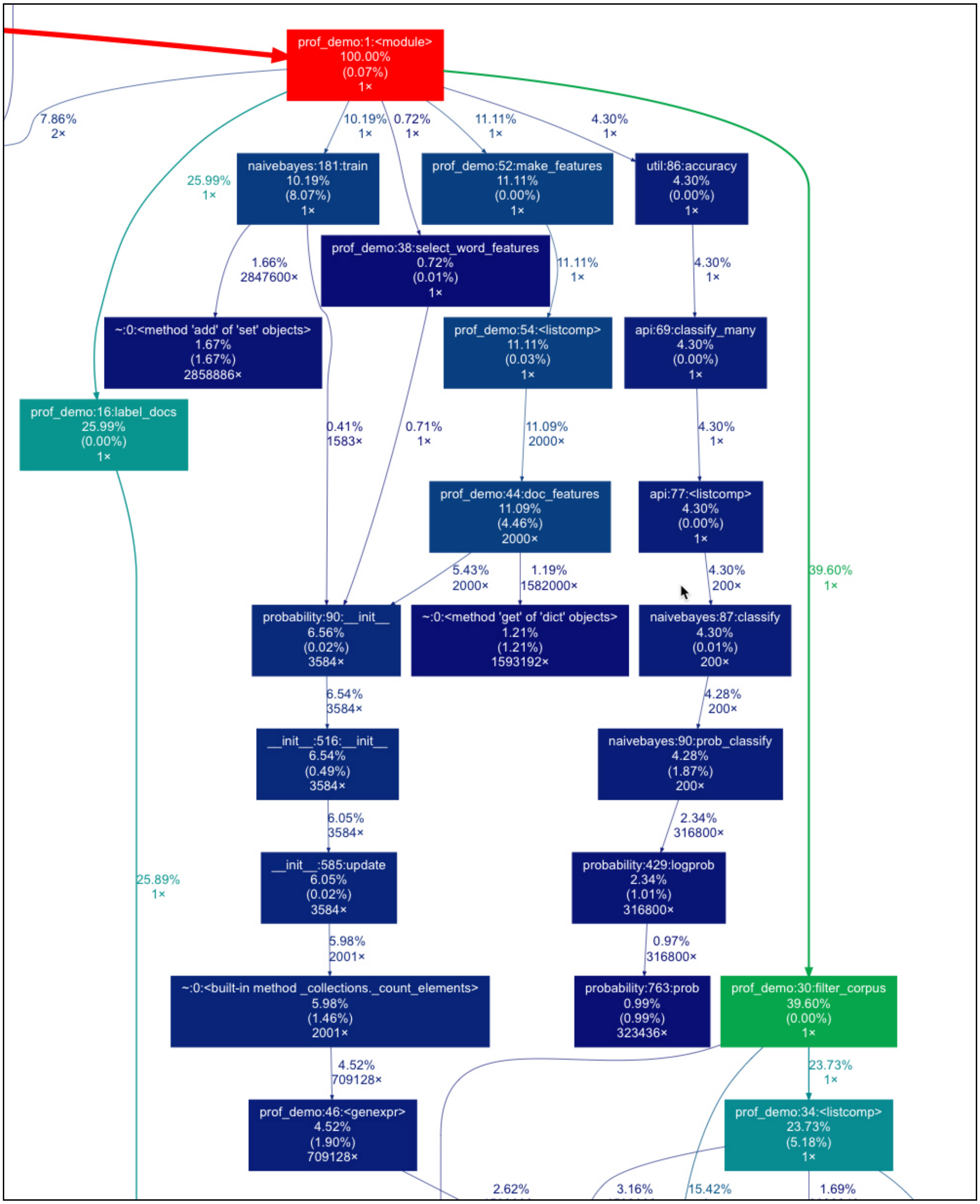


图 12-1

查询分析工具输出结果的方法如下。

```
$ python3 -m pstats /tmp/stat.prof
```

利用这个命令，我们可以把性能分析数据输入到浏览器中。下面我们删除输出结果中的文件名，然后对运行时间进行排序并输出前 10 个数据。

```
/tmp/stat.prof% strip
/tmp/stat.prof% sort time
/tmp/stat.prof% stats 10
```

最终结果如图 12-2 所示。

```
fluid:ch-12 armando$ python3 -m pstats /tmp/stat.prof
Welcome to the profile statistics browser.
/tmp/stat.prof% strip
/tmp/stat.prof% sort time
/tmp/stat.prof% stats 10
Sun Feb  5 18:24:49 2017    /tmp/stat.prof

         30643998 function calls (30123080 primitive calls) in 15.502 seconds

   Ordered by: internal time
   List reduced from 3823 to 10 due to restriction <10>

   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
   319962    2.397    0.000    2.397    0.000 {method 'findall' of '_sre.SRE_Pattern' objects}
        1    1.251    1.251    1.580    1.580 naivebayes.py:181(train)
   319960    1.056    0.000    2.748    0.000 data.py:1114(readline)
  6343280    0.847    0.000    7.338    0.000 util.py:261(iterate_from)
        1    0.803    0.803    3.678    3.678 prof_demo.py:34(<listcomp>)
  3167640    0.741    0.000    0.896    0.000 prof_demo.py:26(isStopWord)
     2000    0.692    0.000    1.719    0.001 prof_demo.py:44(doc_features)
  3167642    0.628    0.000    4.361    0.000 util.py:388(iterate_from)
   371223    0.393    0.000    0.898    0.000 data.py:1353(_read)
3885294/3376152    0.359    0.000    2.696    0.000 {built-in method builtins.len}


/tmp/stat.prof% 
```

图 12-2

表 12-2 对各个标题进行了简要说明。

表 12-2

标题	说明
ncalls	调用次数
tottime	某个函数的总耗时（不包括调用子函数所耗时间）
percall	等于 tottime/ncalls
cumtime	表示该函数及其所有子函数的调用运行的时间，即函数开始调用到返回所经历的时间。这个数值非常准确，即便递归函数也不例外
percall（第二个）	即函数运行一次的平均时间，等于 cumtime/ncalls

使用以下命令可以退出文件浏览器：

```
/tmp/stat.prof% quit
```

安装和运行该分析工具的命令如下。

```
$ pip3 install line_profiler
$ kernprof -l -v prof_demo.py
```

因为详尽的报告过于冗长，所以这里只给出每个函数的摘要信息。当然，这当中有部分重叠内容。

```
Function: label_docs at line 9 Total time: 6.19904 s
Function: isStopWord at line 19 Total time: 2.16542 s
File: prof_demo.py Function: filter_corpus at line 23
Function: select_word_features at line 32 Total time: 4.05266 s
Function: doc_features at line 38 Total time: 12.5919 s
Function: make_features at line 46 Total time: 14.566 s
Function: split_data at line 50 Total time: 3.6e-05 s
```

12.2 安装 Cython

Cython 程序语言实际上充当了 Python 和 C/C++之间的胶水。利用 Cython 工具，可以用 Python 代码生成 C 代码，然后将 C 代码编译成接近于机器语言的二进制代码。cytoolz 软件包提供了许多实用程序，这些程序都是通过将 Python 的 toolz 软件包 Cython 化得到的。以下命令可以用来安装 cython 和 cytoolz。

```
$ pip3 install cython cytoolz
```

进行后续处理前，先来看看经过 Cython 编译处理后的效果。Python 的 timeit 模块可以帮助我们测量时间。下面通过它来对不同的函数进行测度。我们定义下面的函数，它需要的参数为一段代码、一个函数调用以及该段代码的运行次数，具体如下。

```
def time(code, n):
    times = min(timeit.Timer(code, setup=setup).repeat(3, n))

    return round(1000* np.array(times)/n, 3)
```

我们预定义了一个设置控制字符串，其中包含了所需的代码。下列代码摘自本书代码包中的 timeit.py 文件，需要在本地机器上使用 cython_module 进行编译。

```
import timeit
import numpy as np

setup = '''
import nltk
import cython_module as cm
import collections
from nltk.corpus import stopwords
from nltk.corpus import movie_reviews
from nltk.corpus import names
import string
import pandas as pd
import cytoolz

sw = set(stopwords.words('english'))
punctuation = set(string.punctuation)
all_names = set([name.lower() for name in names.words()])
txt = movie_reviews.words(movie_reviews.fileids()[0])

def isStopWord(w):
    return w in sw or w in punctuation

def isStopWord2(w):
    return w in sw or w in punctuation or not w.isalpha()

def isStopWord3(w):
    return w in sw or len(w) == 1 or not w.isalpha() or w in all_names

def isStopWord4(w):
    return w in sw or len(w) == 1

def freq_dict(words):
    dd = collections.defaultdict(int)

    for word in words:
        dd[word] += 1

    return dd

def zero_init():
    features = {}

    for word in set(txt):
        features['count (%s)' % word] = (0)

def zero_init2():
    features = {}
```

```
    for word in set(txt):
        features[word] = (0)

keys = list(set(txt))

def zero_init3():
    features = dict.fromkeys(keys, 0)

zero_dict = dict.fromkeys(keys, 0)

def dict_copy():
    features = zero_dict.copy()
'''

def time(code, n):
    times = min(timeit.Timer(code, setup=setup).repeat(3, n))

    return round(1000* np.array(times)/n, 3)

if __name__ == '__main__':
    print("Best of 3 times per loop in milliseconds")
    n = 10
    print("zero_init ", time("zero_init()", n))
    print("zero_init2", time("zero_init2()", n))
    print("zero_init3", time("zero_init3()", n))
    print("dict_copy ", time("dict_copy()", n))
    print("\n")

    n = 10**2
    print("isStopWord ", time('[w.lower() for w in txt if not
isStopWord(w.lower())]', n))
    print("isStopWord2", time('[w.lower() for w in txt if not
isStopWord2(w.lower())]', n))
    print("isStopWord3", time('[w.lower() for w in txt if not
isStopWord3(w.lower())]', n))
    print("isStopWord4", time('[w.lower() for w in txt if not
isStopWord4(w.lower())]', n))
    print("Cythonized isStopWord", time('[w.lower() for w in txt if not
cm.isStopWord(w.lower())]', n))
    print("Cythonized filter_sw()", time('cm.filter_sw(txt)', n))
    print("\n")
    print("FreqDist", time("nltk.FreqDist(txt)", n))
    print("Default dict", time('freq_dict(txt)', n))
    print("Counter", time('collections.Counter(txt)', n))
    print("Series", time('pd.Series(txt).value_counts()', n))
    print("Cytoolz", time('cytoolz.frequencies(txt)', n))
    print("Cythonized freq_dict", time('cm.freq_dict(txt)', n))
```

以上代码中有多个不同版本的 isStopword()函数，下面是这些函数的运行时间，这里以毫秒（milliseconds）为单位。

```
isStopWord 0.843
isStopWord2 0.902
isStopWord3 0.963
isStopWord4 0.869
Cythonized isStopWord 0.924
Cythonized filter_sw() 0.887
```

为了进行比较，我们还要对 pass 语句的运行时间进行计时。这里，Cython 编译后的 StopWord()函数是基于过滤最严格的 isStopWord3()函数。如果考察 prof_demo.py 中的 doc_features()函数，就会发现我们并没有仔细检查每个特征单词。相反，我们只对文档中的单词和被选为特征的单词感兴趣。因此，其他单词计数可以放心置零。事实上，如果我们把全部的值初始化为 0，然后复制这个字典，那最好不过了。下面这些函数的相应执行时间如下。

```
zero_init 0.61
zero_init2 0.555
zero_init3 0.017
dict_copy 0.011
```

另外一种改进性能的方法是，使用 Python 内置的 defaultdict 类，而非 NLTK 提供的 FreqDist 类。相应程序运行时间如下。

```
FreqDist 2.206
Default dict 0.674
Counter 0.79
Series 7.006
Cytoolz 0.542
Cythonized freq_dict 0.616
```

就像我们看到的那样，Cython 编译后的版本始终是要快一些，尽管有时候不是快很多。

12.3　调用 C 代码

我们可以从 Cython 调用 C 函数。C 语言的字符串函数 strlen()相当于 Python 语言中的 len()函数。当从一个 Cython 的.pyx 文件中调用这个 C 函数时，需要将其导入，代码如下。

```
from libc.string cimport strlen
```

这样，我们就可以在.pyx 文件的其他地方来调用这个 strlen()函数了。而这个.pyx 文件可

以随意包含任何的 Python 代码。下面的代码摘自本书代码包中的 cython_module.pyx 文件。

```
from collections import defaultdict
from nltk.corpus import stopwords
from nltk.corpus import names
from libc.string cimport strlen

sw = set(stopwords.words('english'))
all_names = set([name.lower() for name in names.words()])

def isStopWord(w):
     py_byte_string = w.encode('UTF-8')
     cdef char* c_string = py_byte_string
     truth = (w in sw) or (w in all_names) or (not w.isalpha()) or
(strlen(c_string) == 1)
     return truth

def filter_sw(words):
    return [w.lower() for w in words if not isStopWord(w.lower())]

def freq_dict(words):
    dd = defaultdict(int)

    for word in words:
        dd[word] += 1

    return dd
```

编译这段代码，我们需要一个包含以下内容的 setup.py 文件。

```
from distutils.core import setup
from Cython.Build import cythonize

setup(
    ext_modules = cythonize("cython_module.pyx")
)
```

编译程序代码的命令如下。

```
$ python3 setup.py build_ext --inplace
```

以下输出显示了 cython_module 扩展的构建信息。

```
$python3 setup.py build_ext --inplace
```

```
running build_ext
building 'cython_module' extension
creating build
creating build/temp.macosx-10.12-x86_64-3.6
clang -Wno-unused-result -Wsign-compare -Wunreachable-code -fno-common -
dynamic -DNDEBUG -g -fwrapv -O3 -Wall -Wstrict-prototypes -
I/usr/local/include -I/usr/local/opt/openssl/include -
I/usr/local/opt/sqlite/include -
I/usr/local/Cellar/python3/3.6.0/Frameworks/Python.framework/Versions/3.6/i
nclude/python3.6m -c cython_module.c -o build/temp.macosx-10.12-
x86_64-3.6/cython_module.o
clang -bundle -undefined dynamic_lookup build/temp.macosx-10.12-
x86_64-3.6/cython_module.o -L/usr/local/lib -L/usr/local/opt/openssl/lib -
L/usr/local/opt/sqlite/lib -o
/Users/armando/gdrive/projects/bitbucket/pubs/2016-pda-e2-
packt/chapters/ch-12/cython_module.cpython-36m-darwin.so
```

如今，我们可以修改这个情感分析程序，让它来调用 Cython 函数。此外，我们还可以根据前面曾经介绍的方法来改善这个代码的性能。因为一些函数会重复使用，所以我们将这些函数抽取出来并集中到本书代码包中的 core.py 文件中。下面的代码摘自本书代码包中的 cython_demo.py 文件（这些代码需要本地机器能够支持 cython_module）。

```
from nltk.corpus import movie_reviews
from nltk import NaiveBayesClassifier
from nltk.classify import accuracy
import cython_module as cm
import cytoolz
from core import label_docs
from core import filter_corpus
from core import split_data
def select_word_features(corpus):
    words = cytoolz.frequencies(filtered)
    sorted_words = sorted(words, key=words.get)
    N = int(.02 * len(sorted_words))

    return sorted_words[-N:]

def match(a, b):
    return set(a.keys()).intersection(b)

def doc_features(doc):
    doc_words = cytoolz.frequencies(cm.filter_sw(doc))

    # initialize to 0
    features = zero_features.copy()
```

```
        word_matches = match(doc_words, word_features)

        for word in word_matches:
            features[word] = (doc_words[word])

        return features

    def make_features(docs):
        return [(doc_features(d), c) for (d,c) in docs]

    if __name__ == "__main__":
        labeled_docs = label_docs()
        filtered = filter_corpus()
        word_features = select_word_features(filtered)
        zero_features = dict.fromkeys(word_features, 0)
        featuresets = make_features(labeled_docs)
        train_set, test_set = split_data(featuresets)
        classifier = NaiveBayesClassifier.train(train_set)
        print("Accuracy", accuracy(classifier, test_set))
        print(classifier.show_most_informative_features())
```

我们使用 time 命令执行代码。

```
$ time python3 cython_demo.py
```

表 12-3 是对 time 命令运行结果的总结。注意，括号中的值是最小值。

表 12-3

时间类型	第一轮	第二轮	第三轮
real	(9.639)	9.817	9.912
user	(9.604)	9.661	9.683
sys	(0.404)	0.424	0.451

与以前的代码的执行时间相比，我们可以看到性能有了明显的提升。时间表 12-4 是从 12.2 节中转载过来的，以便于进行比较。

表 12-4

时间类型	第一轮	第二轮	第三轮
real	11.521	10.808	(10.416)
user	9.758	9.826	(9.444)
sys	0.965	0.643	(0.620)

12.4　利用 multiprocessing 创建进程池

multiprocessing 是 Python 的一个标准模块，可以用于多处理器机器。multiprocessing 通过创建多个进程，成功解决了全局解释器锁（the Global Interpreter Lock，GIL）问题。

> **提示**
> GIL 会锁定 Python 的字节码，导致只有一个线程可以访问这些字节码。

multiprocessing 支持进程池、队列和管道技术。进程池实际上就是可以并行执行一个函数的一组系统进程。队列是一些数据结构，通常用于存储任务。管道用来连接不同的进程，而且连接方式为一个进程的输出作为另一个进程的输入。

> **提示**
> Python 的 Windows 平台版本没有实现 os.fork()函数，因此我们务必确保导入该函数并且将 def 语句块定义在 if __name__ == "__main__"语句块之外。

下面创建一个进程池并注册一个函数，代码如下。

```
p = mp.Pool(nprocs)
```

进程池有一个 map()方法，我们可以看成是 Python 并行的 map()函数。

```
p.map(simulate, [i for i in xrange(10, 50)])
```

模拟微粒的一维运动，它实际上进行的是随机游走，我们这里关心的是微粒终点位置的均值。我们重复这个模仿实验，每次具有不同的步长。实际上，计算本身并不重要，重要的是与单个进程相比，多个进程的加速效果如何，我们将通过 matplotlib 绘制加速比。完整的代码摘自本书代码包中的 multiprocessing_sim.py 文件。

```
from numpy.random import random_integers
from numpy.random import randn
import numpy as np
import timeit
```

```
import argparse
import multiprocessing as mp
import matplotlib.pyplot as plt

def simulate(size):
    n = 0
    mean = 0
    M2 = 0

    speed = randn(10000)

    for i in range(1000):
        n = n + 1
        indices = random_integers(0, len(speed)-1, size=size)
        x = (1 + speed[indices]).prod()
        delta = x - mean
        mean = mean + delta/n
        M2 = M2 + delta*(x - mean)

    return mean

def serial():
    start = timeit.default_timer()

    for i in range(10, 50):
        simulate(i)

    end = timeit.default_timer() - start
    print("Serial time", end)

    return end
def parallel(nprocs):
    start = timeit.default_timer()
    p = mp.Pool(nprocs)
    print(nprocs, "Pool creation time", timeit.default_timer() - start)

    p.map(simulate, [i for i in range(10, 50)])
    p.close()
    p.join()

    end = timeit.default_timer() - start
    print(nprocs, "Parallel time", end)
    return end

if __name__ == "__main__":
```

```
ratios = []
baseline = serial()

for i in range(1, mp.cpu_count()):
    ratios.append(baseline/parallel(i))

plt.xlabel('# processes')
plt.ylabel('Serial/Parallel')
plt.plot(np.arange(1, mp.cpu_count()), ratios)
plt.grid(True)
plt.show()
```

当进程池大小由 1 变到 8 时，加速比的变化如图 12-3 所示。

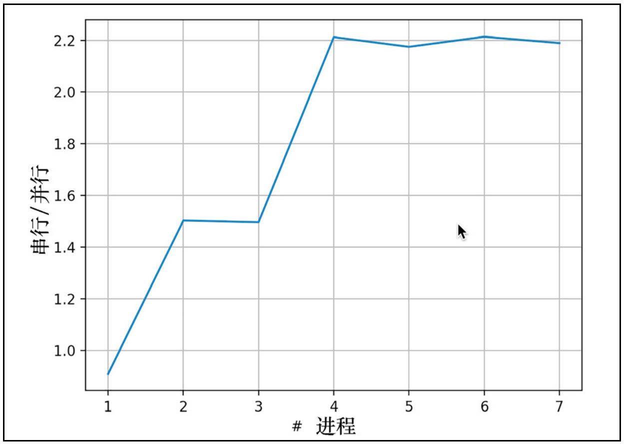


图 12-3

阿姆达尔定律（详情请访问 http://en.wikipedia.org/wiki/Amdahl%27s_law）代表了处理器平行运算后效率提升的能力。这个定律可以用来预测加速比的最大可能值。进程的数量限制了加速比的绝对最大值。不过，图 12-3，使用双进程时速度并没有加倍，使用 3 个进程时速度也没有比原来快 3 倍，但是可以接近这个方向。任何给定的 Python 代码，总会有些部分是无法并行化的。例如，我们可能需要等待资源被释放，或者进行的计算必须串行完成。有时，我们还必须考虑并行化配置和进程间相应通信带来的开销。阿姆达尔定律指出，加速比的倒数、进程数量的倒数和代码中无法并行计算部分所占的比例之间存在线性关系。

12.5 通过 Joblib 提高 for 循环的并发性

Joblib 是一个由 scikit-learn 的开发者创建的 Python 库，旨在改善长时间运行的 Python

函数的性能。实际上，Joblib 是通过在幕后使用多进程或者线程技术来实现高速缓存和并行化从而达到提升性能的目的。安装 Joblib 的方法如下。

```
$ pip3 install joblib
```

我们会重新使用前面的示例代码，只是变更 parallel()函数。相关的代码摘自本书代码包中的 joblib_demo.py 文件。

```
def parallel(nprocs):
    start = timeit.default_timer()
    Parallel(nprocs)(delayed(simulate)(i) for i in xrange(10, 50))

    end = timeit.default_timer() - start
    print(nprocs, "Parallel time", end)
    return end
```

最终结果如图 12-4 所示（注意，实际的处理器数量取决于我们的硬件）。

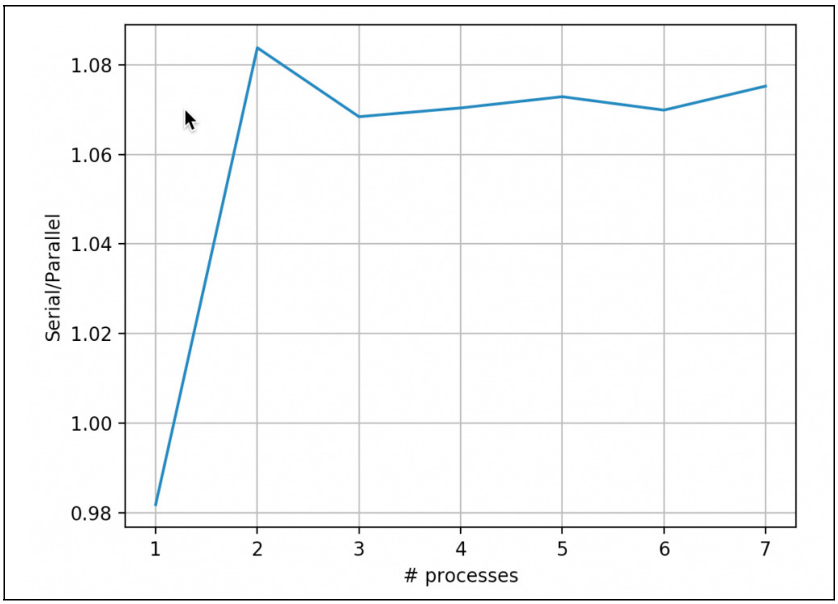


图 12-4

12.6 比较 Bottleneck 函数与 NumPy 函数

Bottleneck 是受到 Numpy 和 Scipy 的启发而创建的一组函数，它们着眼于高性能，都

是由 Cython 写成的。Bottleneck 为数组维数、坐标轴和数据类型的每种组合都提供了单独的 Cython 函数。但是，这并不表示最终用户和 Bottleneck 的限制因素决定了到底执行哪一个 Cython 函数。安装 Bottleneck 的命令如下。

```
$ pip3 install bottleneck
```

下面对 numpy.median()和 SciPy.stats 的执行时间进行比较。

这对人为决定是否使用 Cython 函数很有帮助，尤其是在将这些 Cython 函数用于紧密循环（tight loop）或者频繁调用的函数中时。下面打印 Bottleneck 中 median()函数的名称，代码如下。

```
import bottleneck as bn
import numpy as np
import timeit

setup = '''
import numpy as np
import bottleneck as bn
from scipy.stats import rankdata

np.random.seed(42)
a = np.random.randn(30)
'''
def time(code, setup, n):
    return timeit.Timer(code, setup=setup).repeat(3, n)

if __name__ == '__main__':
    n = 10**3
    print(n, "pass", max(time("pass", "", n)))
    print(n, "min np.median", min(time('np.median(a)', setup, n)))
    print(n, "min bn.median", min(time('bn.median(a)', setup, n)))
    a = np.arange(7)
    print)"Median diff", np.median(a) - bn.median(a))

    print(n, "min scipy.stats.rankdata", min(time('rankdata(a)', setup,
n)))
    print(n, "min bn.rankdata", min(time('bn.rankdata(a)', setup, n)))
```

下面是函数名和相应的运行时间。

```
$ python3 bn_demo.py
1000 pass 7.228925824165344e-06
1000 min np.median 0.019842895912006497
```

```
1000 min bn.median 0.0003261091187596321
Median diff 0.0
1000 min scipy.stats.rankdata 0.04070987505838275
1000 min bn.rankdata 0.0011222949251532555
```

显然，Bottleneck 快极了。不过，Bottleneck 提供的函数还不多。表 12-5 给出了已实现的函数。

表 12-5

种类	函数
NumPy/SciPy	median、nanmedian、rankdata、ss、nansum、nanmin、nanmax、nanmean、nanstd、nanargmin 和 nanargmax
函数	nanrankdata、nanvar、partsort、argpartsort、replace、nn、anynan 和 allnan
滑动窗口	move_sum、move_nansum、move_mean、move_nanmean、move_median、move_std、move_nanstd、move_min、move_nanmin、move_max 和 move_nanmax

12.7 通过 Jug 实现 MapReduce

Jug 是一个分布式计算框架，它以任务作为并行化的主要单位。作为后端程序，Jug 要用到文件系统或者 Redis 服务器。Redis 服务器已经在第 8 章中介绍过，下面我们介绍 Jug 的安装方法，命令如下。

```
$ pip3 install jug
```

MapReduce 是一种分布式算法，它可以通过计算机集群来处理大规模数据。这个算法通常包含映射和化简两个步骤。映射阶段，数据是以并行方式进行处理的。这时，数据将被划分成一些数据块，而且过滤及其他操作都是针对每个数据块进行的。化简阶段对映射阶段的处理结果进行合并，如创建一个统计报告。

如果有一个文本文件列表，那么我们可以针对每个文件来计算单词计数，这个工作可以在映射阶段完成。最后，我们可以把各个单词计数组合成一个语料库词频字典。Jug 提供了 MapReduce 功能，我们可以通过本书代码包中的 jug_demo.py 文件加以演示。需要注意的是，这段代码依赖于 cython_module。

```
import jug.mapreduce
```

```
from jug.compound import CompoundTask
import cython_module as cm
import cytoolz
import pickle

def get_txts():
    return [(1, 'Lorem ipsum dolor sit amet, consectetur adipiscing
elit.'), (2, 'Donec a elit pharetra, malesuada massa vitae, elementum
dolor.'), (3, 'Integer a tortor ac mi vehicula tempor at a nunc.')]

def freq_dict(file_words):
    filtered = cm.filter_sw(file_words[1].split())

    fd = cytoolz.frequencies(filtered)

    return fd

def merge(left, right):
    return cytoolz.merge_with(sum, left, right)

merged_counts = CompoundTask(jug.mapreduce.mapreduce, merge, freq_dict,
get_txts(), map_step=1)
```

上述代码在化简阶段调用了 merge()函数，在映射阶段调用了 freq_dict()函数。此外，我们还定义了一个包含多个子任务的 Jug CompoundTask。运行这段代码前，需要启动一个 Redis 服务器。然后，就可以利用以下命令进行 MapReduce 处理了。

```
$ jug execute jug_demo.py --jugdir=redis://127.0.0.1/&
```

以上命令末尾的&符号表示这条命令将在后台运行。利用这种方式，可以从多台计算机来执行该命令，只要 Redis 服务器可以通过访问即可。在这个例子中，Redis 仅在本地计算机上运行，本地主机的 IP 地址是 127.0.0.1。但是，我们仍然可以在本机上多次运行该命令。我们可以查看该 Jug 命令的状态，方法如下。

```
$ jug status jug_demo.py
```

默认情况下，如果没有设置 jugdir 参数，Jug 会将数据存放到当前工作目录下。如果要清除 Jug 相关目录，我们可以使用如下命令。

```
$ jug cleanup jug_demo.py
```

如果想查询 Redis 并进行其他分析工作，还需要用到其他程序。

在这个程序中，使用下列命令来初始化 Jug。

```
jug.init('jug_demo.py', 'redis://127.0.0.1/')
import jug_demo
```

以下代码用来获得化简阶段生成的结果。

```
words = jug.task.value(jug_demo.merged_counts)
```

以下代码取自本书代码包中的 jug_redis.py 文件。

```
import jug

def main():
    jug.init('jug_demo.py', 'redis://127.0.0.1/')
    import jug_demo
    print("Merged counts", jug.task.value(jug_demo.merged_counts))

if __name__ == "__main__":
    main()
```

12.8 安装 MPI for Python

消息传递接口（The Message Passing Interface，MPI）是一种标准协议，由计算机专家开发用来实现分布式计算机的广泛协作。20 世纪 90 年代，MPI 主要用于 Fortran 和 C 语言编写的程序中，但是，它不依赖具体的硬体和计算机语言。MPI 函数可以完成发送和接收操作、实现 MapReduce 功能和同步。MPI 既有处理两个处理器的点对点函数，也提供了处理所有处理器之间的操作的相应函数。MPI 支持多种计算机语言，也就是说，提供了多种语言绑定，其中就包括 Python。读者可以自行下载 MPI，在本书写作时，MPI 的最新版本为 2.0.2。当然，我们也可以通过网络检查是否有更新的版本。安装 MPI 的时候时间可能有点长，大概需要 30min。下面是具体的安装命令，这里假设将其安装到 /usr/local 目录下面。

```
$ ./configure --prefix=/usr/local
$ make all
$ sudo make install
```

下面来安装 MPI 的 Python 绑定，命令如下。

```
$ pip3 install mpi4py
```

12.9 IPython Parallel

IPython Parallel 是用于并行计算的 IPython 应用程序接口。这里，我们用它来通过 MPI 来传递消息。为此，我们可能需要设置相应的环境变量，命令如下。

```
$ export LC_ALL=en_US.UTF-8
$ export LANG=en_US.UTF-8
```

在命令行执行以下命令。

```
$ ipython3 profile create --parallel --profile=mpi
```

以上命令将在我们的 home 目录下面创建文件，具体路径为.ipython/ profile_mpi/。

启动一个使用 MPI 性能分析的集群，命令如下。

```
$ ipcluster start --profile=mpi --engines=MPI --debug
```

上面的命令中，规定性能分析工具为 mpi，同时指定 MPI 引擎提供调试级别的记录功能。这样，我们就可以通过 IPython Notebook 与集群进行交互了。输入下列命令，我们得到一个具有绘图功能的笔记本（notebook），同时 NumPy、SciPy 和 matplotlib 会自动导入。

```
$ jupyter-notebook --profile=mpi --log-level=DEBUG
```

以上命令使用 mpi 作为调试记录级别的性能分析工具。对于这个笔记本例子而言，它存放于本书代码包中的 IPythonParallel.ipynb 文件中。下面导入 IPython Parallel 的 Client 类和 statsmodels.api 模块，命令如下。

```
In [1]:from ipyparallel import Client
import statsmodels.api as sm
```

下面加载太阳黑子数据，然后计算平均值。

```
In [2]: data_loader = sm.datasets.sunspots.load_pandas()
vals = data_loader.data['SUNACTIVITY'].values
glob_mean = vals.mean()
glob_mean
```

以下是输出结果。

```
Out [2]: 49.752103559870541
```

创建一个客户端，命令如下。

```
In [3]: c = Client(profile='mpi')
```

为客户端创建一个视图，命令如下。

```
In [4]: view=c[:]
```

IPython 提供了许多魔术（magics）命令，即 IPython notebooks 的一些专用的命令。如果要启用这些命令，方法如下。

```
In [5]: view.activate()
```

下面来加载摘自本书代码包中的 mpi_ipython.py 文件。

```
from mpi4py import MPI
from numpy.random import random_integers
from numpy.random import randn
import numpy as np
import statsmodels.api as sm
import bottleneck as bn
import logging

def jackknife(a, parallel=True):
    data_loader = sm.datasets.sunspots.load_pandas()
    vals = data_loader.data['SUNACTIVITY'].values

    results = []

    for i in a:
        tmp = np.array(vals.tolist())
        tmp[i] = np.nan
        results.append(bn.nanmean(tmp))
    results = np.array(results)

    if parallel:
        comm = MPI.COMM_WORLD
        rcvBuf = np.zeros(0.0, 'd')
        comm.gather([results, MPI.DOUBLE], [rcvBuf, MPI.DOUBLE])

    return results
```

```
if __name__ == "__main__":
    skiplist = np.arange(39, dtype='int')
    print(jackknife(skiplist, False))
```

上面的程序中包含了一个执行刀切法重采样的函数。刀切法重采样技术是一种重采样技术，即先删除样本中的一个观测值，然后再根据需要进行相应的统计估计。就本例而言，我们关心的是平均值。剔除一个观测值的方法是：将其置为 NumPy NaN，然后对新样本执行 Bottleneck 的 nanmean()函数。下面先来执行加载任务。

```
In [6]: view.run('mpi_ipython.py')
```

然后，拆取并扩展一个数组，让它带有太阳黑子数组全部索引。

```
In [7]: view.scatter('a',np.arange(len(vals),dtype='int'))
```

数组 a 可以在笔记本中显示出来，命令如下。

```
In [8]: view['a']
```

以上命令的输出结果如下。

```
Out[8]:[array([ 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14,
15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32,
33, 34, 35, 36, 37, 38]), ... TRUNCATED ...]
```

在所有客户端上调用 jackknife()函数。

```
In [9]: %px means = jackknife(a)
```

所有工作进程（worker processes）结束后，就可以查看结果了。

```
In [10]: view['means']
```

结果是一个列表，长度与我们启动的进程数量相同。每个进程都返回一个 NumPy 数组，其中存放刀切法重采样技术计算得到的平均值。这个结构不是十分有用，因此将其转换为一个比较宽的列表。

```
In [11]: all_means = []

for v in view['means']:
    all_means.extend(v)
```

```
mean(all_means)
```

这时输出结果如下。

```
Out [11]: 49.752103559870577
```

我们还可以计算标准差，不过这个太简单，这里就不多解释了。下面，我们不妨将刀切法求出的平均值用直方图画出来。

```
In [13]: hist(all_means, bins=sqrt(len(all_means)))
```

进行故障排除时，我们可以使用以下命令来显示工作进程的错误信息。

```
In [14]: [(k, c.metadata[k]['started'], c.metadata[k]['pyout'],
c.metadata[k]['pyerr']) for k in c.metadata.keys()]
```

12.10 小结

本章对摘自第 9 章中的情感分析脚本进行了性能调优。我们通过性能分析、Cython 和多种性能提升措施，最终让它的速度翻倍了。此外，我们还使用 multiprocessing、Joblib、Jug 和 MPI via IPython Parallel 充分利用了并行化。

本章是本书的最后一章，当然，我们学习的步伐不会就此停住，我们需要不停地修改代码，直到能够满足要求为止。如果我们有一个专门的数据分析项目，那再好不过了，即使仅仅是一个用于练手的项目。如果还没有，我们不妨参加竞赛活动。许多竞赛都提供了不错的奖品。

附录 A 重要概念

本附录简要回顾本书中涉及的技术术语和概念。

阿姆达尔定律可以推算并行化带来的加速比的最大可能取值。因为受到进程数量的限制，所以加速比的绝对最大值不可能无限大。任何给定的 Python 代码中，总有某些部分代码无法实现并行化。此外，我们还必须考虑并行化设置和有关进程间通信所引起的开销。阿姆达尔定律指出，加速比倒数、进程数量倒数以及无法并行化的程序代码所占比重之间存在线性相关。

ARMA 模型是自回归模型和移动平均模型的组合体，常用于预测时间序列的未来值。

人工神经网络（ANN）是受到生物大脑的启发，用神经元构成的、带有输入端和输出端的一类网络。神经元的输出，可以继续传递给神经元作为输入，以此类推，便可以得到一个多层网络。神经网络含有自适应元件（adaptive elements），因此，它们非常适合用来处理非线性模型以及模式识别问题。

扩展的迪克-富勒（Augmented Dickey-Fuller，ADF）检验，常称 ADF 检验，是一种针对协整关系的统计检验。ADF 检验常用于测试时间序列的稳定性。

自相关指的是同一个数据集不同时刻的取值之间的相关程度，通常用来表示趋势。例如，如果后延一个周期，这时就可以检查前值是否对现值有影响。如果有影响，那么自相关值必定是相当高的。

自相关图可以用来描绘时间序列数据在不同时间延迟下的自相关程度。自相关是具有相同时滞的时间序列之间的相关程度。

自回归模型是一种利用时间序列中前面的值，通过（一般为线性）回归来预测将来值的模型。自回归模型是 ARMA 模型的一个特例，相当于具有零移动平均分量的 ARMA 模型。

词袋模型是一个简化的文本模型，其中文本由一袋单词表示。对于这种表示方法，单词的次序忽略不计。一般情况下，这个模型把单词计数（word counts）或者说是某个单词的存在性作为其特征。

泡式图是一种扩展的散点图，其中，第 3 个变量的值是通过包围数据点的气泡的大小来表示的。

Cassandra 查询语言（Cassandra Query Language，CQL）是一种用来查询 Apache Cassandra 的语言，其语法与 SQL 类似。

协整类似于相关性，也是时间序列数据的一个统计特性。协整通常用来衡量两个时间序列之间的同步程度。

聚类分析的目的是把数据分成不同的组，每个组称为一个聚类。如果训练数据未经标记，这时聚类分析就属于无监督的分析方法。一些聚类算法要求推测聚类数，另一些算法则没有这种要求。

层叠样式表（Cascading Style Sheets，CSS）是一种描述网页样式元素的语言。它是由万维网协会（the World Wide Web Consortium）开发和维护的。

所谓 **CSS 选择器**，实际上就是一些用于选择网页内容的规则。

NumPy 的**字符码**用来向后兼容 **Numeric**，因为 Numeric 是 NumPy 的前身。

数据类型对象，实际上就是 `numpy.dtype` 类的各种实例。这些数据类型对象通常都提供了面向对象的接口，以供操作 NumPy 的数据类型。

特征值是方程 `Ax = ax` 的标量解，其中 A 是一个二维矩阵，而 x 是一个一维向量。

特征向量（eigenvectors）用来表示特征值的向量。

指数移动平均法是一种权值随时间以指数形式递减的移动平均方法。

快速傅里叶变换（Fast Fourier Transform，FFT）是一种计算傅里叶变换的快捷方法。FFT 的计算复杂度为 O(N log N)，这对于以前的算法而言，性能得到了极大的提升。

过滤是一种信号处理技术，它涉及对信号的某些部分进行删减或抑制。过滤的类型有很多，其中包括中值和 Wiener 滤波。

傅里叶分析是建立在以数学家 Joseph Fourier 命名的傅里叶级数之上的一种数学方法。傅里叶级数是一种表示函数的数学方法，它通常使用正弦函数和余弦函数构成的无穷级数来表示函数。当然，这里的函数既可以是实值函数，也可以是复值函数。

遗传算法是一种基于生物进化论的搜索和优化算法。

图形处理单元（GPU）是专门用于高效显示图像的集成电路。近来，GPU 已经开始用于完成大规模并行计算，如训练神经网络等。

层次数据格式（Hierarchical Data Format，HDF）是一种存储大型数值数据的技术规范。同时，HDF 工作组还专门为这种规范提供了一个软件库。

希尔伯特-黄变换（Hilbert-Huang Transform）是一种分解信号的数学算法，这个方法可以用于发现时间序列数据中的周期循环。它已经成功用于确定太阳黑子活动的周期。

超文本标记语言（HyperText Markup Language，HTML）是创建 Web 页面的基础性技术，它为媒体、文本和超链接定义了相应的标签。

国际互联网工程任务组（Internet Engineering Task Force，IETF）是一个致力于维护和开发因特网的开放式工作组。这里的开放，是指任何人都可以参与规则的制定。

JavaScript 对象表示法（JavaScript Object Notation，JSON）是一种数据格式，利用这种格式，我们就可以使用 JavaScript 表示法来表示数据了。相对于诸如 XML 之类的数据格式来说，JSON 更加简洁。

K-折交叉验证是一种交叉验证形式，其中，它会把数据集随机分为 *k*（一个小整数）份，每一份称为一个**包**。在 *k* 次迭代中，每个包被用作一次验证，其他时候用于训练，最后，对迭代结果进行合并处理。

Kruskal-Wallis 单因子方差分析是一种统计学方法，它能够在没有对总体分布做出假定的情况下对样本方差进行相应的统计分析。

时滞图是一种描绘某个时间序列及其时滞序列的散点图，它为我们展示了具有时滞的时间序列数据与原来序列之间的自相关性。

学习曲线是一种展示学习算法行为特点的可视化方法，用来勾画训练成效和测试成效是如何随训练数据量的变化而变化的。

对数图是一种对数坐标图形，这类图形在数据变化巨大的情况下非常有用，因为图形显示的是数量级。

逻辑回归是一种分类算法，可以用于预测属于某类别或者某事件发生的概率。逻辑回归是以 logistic 函数为基础的，该函数的取值范围为 0～1，正好与概率的取值范围吻合。因此，我们可以使用 logistic 函数把任意值转换为概率值。

MapReduce 是一种分布式算法，可利用计算机集群来处理大规模数据。该算法通常需

要经历映射和化简两个阶段。在映射阶段，数据是以并行的方式进行处理的。这时，数据将被划分成一些数据块，而且过滤及其他操作都是针对每个数据块进行的。化简阶段就是对映射阶段的结果进行合并处理。

摩尔定律实际上是观察到的一种现象，即现代电脑芯片上的晶体管的数目每两年就翻一番。从 1970 年以来，这个趋势一直保持至今。此外，还有一个第二摩尔定律，也就是著名的 Rock 定律。这个定律指出，集成电路的研发和制造成本正在呈指数式增长。

移动平均法指定向前所能看到的数据的窗口大小，而且该窗口每次前移一个周期的时候，都会计算其平均值。移动平均法的类型有很多种，它们的区别主要在于求平均值所用的权重有所不同。

朴素贝叶斯分类是一种基于概率与数理统计领域中的贝叶斯定理的概率分类算法，它之所以称为朴素，是因为它假设属性之间相互独立。

对象关系映射（Object-Relational Mapping，ORM）是一种软件体系结构模式，用来实现数据库模式和面向对象计算机语言之间的转换。

观点挖掘或者情感分析是一个研究领域，旨在有效发现和评估文本内的意见和情绪。

词性（Part of Speech，POS）标签是加注于句子中每个单词上的各种标签，这些标签都具有相应的语法含义，如动词或者名词等。

表述性状态转移（Representational State Transfer，REST）是一种网络服务架构风格。

简易信息聚合（Really Simple Syndication，RSS）是一种诸如博客之类的网络订阅源（Web Feeds）的公布和检索标准。

散点图是一种二维图像，用来展示直角坐标系中两个变量之间的关系，其中，一个变量值表示它在一个坐标轴的坐标，另一个变量值表示它在另一个坐标轴的坐标。这样，我们就可以迅速地绘制其相互关系了。

信号处理是隶属于工程和应用数学的一个领域，对模拟信号和数字信号进行处理分析。当然，这里的模拟信号和数字信号可以看作是随时间变化的一些变量。

SQL 是一种关系数据库查询和操作的专用语言，可以用来创建数据表、向数据表插入数据行以及删除数据表等。

停用词指的是那些常用但是信息含量很低的字词。进行文本分析前，我们通常需要将停用词先删除掉。尽管过滤停用词是一个惯例，但是对于停用词至今尚未有标准的定义。

监督学习是一种机器学习技术，它要求使用带有标签的训练数据。

支持向量机（SVM）可以用来完成回归（SVR）和分类（SVC）任务。SVM 会将数据点映射到一个多维空间，这个映射过程通常由所谓的核函数完成。核函数可以是线性的，也可以是非线性的。

词频和逆文档频率（TF-IDF）是衡量语料库中的单词重要性的一种度量指标。它通常包括单词出现的频率和逆文档频率。词频用来表示单词在一个文档中出现的次数。对于逆文档频率，我们先求出其中含有该单词的文档数量，然后取其倒数即可。

时间序列是对数据点按照取样时间先后顺序排列的一个有序列表。通常，每个数据点都带有一个相应的时间戳。时间序列可能是稳定的，也可能是不稳定的。

附录 B
常用函数

本附录列出了一些常用函数并根据其所属程序包进行组织，这些程序包涉及 matplotlib、NumPy、Pandas、scikit-learn 和 SciPy。

Matplotlib

下面是常用的 matplotlib 函数。

- `matplotlib.pyplot.axis(*v, **kwargs)`：该函数用于获取或者设置坐标轴属性。例如，`axis('off')`表示关闭坐标轴线及其标签。
- `matplotlib.pyplot.figure(num=None, figsize=None, dpi=None, facecolor=None, edgecolor=None, frameon=True, FigureClass=<class 'matplotlib.figure.Figure'>, **kwargs)`：这个函数用于新建一个图像。
- `matplotlib.pyplot.grid(b=None, which='major', axis='both', **kwargs)`：这个函数用于打开或者关闭图像中的网格线。
- `matplotlib.pyplot.hist(x, bins=10, range=None, normed=False, weights=None, cumulative=False, bottom=None, histtype='bar', align='mid', orientation ='vertical', rwidth=None, log=False, color=None, label=None, stacked=False, hold=None, **kwargs)`：这个函数用于绘制直方图。
- `matplotlib.pyplot.imshow(X, cmap=None, norm=None, aspect=None, interpolation =None, alpha=None, vmin=None, vmax=None, origin= None, extent=None, shape=None, filternorm=1, filterrad =4.0,`

imlim=None, resample=None, url=None, hold=None, **kwargs)：这个函数用来绘制类似数组之类的数据的图像。

- matplotlib.pyplot.legend(*args, **kwargs)：这个函数可以用来在指定位置显示图例或注释性文字，如 plt.legend(loc='best')。
- matplotlib.pyplot.plot(*args, **kwargs)：这个函数可以通过单个或多个（*x*，*y*）坐标以及相应可选的格式串来创建二维图像。
- matplotlib.pyplot.scatter(x, y, s=20, c='b', marker='o', cmap=None, norm=None, vmin=None, vmax=None, alpha=None, linewidths =None, verts=None, hold=None, **kwargs)：这个函数可以为两个数组创建散点图。
- matplotlib.pyplot.show(*args, **kw)：这个函数用来显示图像。
- matplotlib.pyplot.subplot(*args, **kwargs)：在给定行号、列号和图形索引号的情况下，这个函数可以用来创建子图。所有的这些数字都是从 1 开始计数。例如，plt.subplot(221)的作用是创建 2×2 网格中的第一个子图。
- matplotlib.pyplot.title(s, *args, **kwargs)：这个函数的作用是给图像加标题。

NumPy

下面是常用的 NumPy 函数。

- Numpy.arange([start,] stop[, step,], dtype=None)：这个函数可以利用指定范围内的等差数值来创建 NumPy 数组。
- Numpy.argsort(a, axis=-1, kind='quicksort', order=None)：这个函数将返回一个与数组元素将来排列顺序一致的下标序列。
- Numpy.array(object, dtype=None, copy=True, order=None, subok =False, ndmin=0)：这个函数可以从类似数组的序列，例如 Python 列表来创建 NumPy 数组。
- Numpy.dot(a, b, out=None)：这个函数将计算两个数组的点积。
- Numpy.eye(N, M=None, k=0, dtype=<type 'float'>)：这个函数将返回单位矩阵。

- Numpy.load(file, mmap_mode=None)：这个函数用来从.npy、.npz 或者 pickle 中加载 NumPy 数组或者经过序列化处理的对象。内存映射数组（memory-mapped array）将被存放到文件系统中并且不必全部载入内存，这对大型数组来说非常有用。
- Numpy.loadtxt(fname, dtype=<type 'float'>, comments='#', delimiter= None, converters=None, skiprows=0, usecols=None, unpack=False, ndmin=0)：这个函数可以把文本文件中的数据加载到一个 NumPy 数组中。
- Numpy.mean(a, axis=None, dtype=None, out=None, keepdims =False)：这个函数可以计算给定坐标轴上坐标值的算术平均值。
- Numpy.median(a, axis=None, out=None, overwrite_input=False)：这个函数可以用来计算给定坐标轴上坐标值的中值。
- Numpy.ones(shape, dtype=None, order='C')：这个函数可以用来创建指定形状和数据类型的 NumPy 数组，其元素初始值为 1。
- Numpy.polyfit(x, y, deg, rcond=None, full=False, w=None, cov=False)：这个函数可以用来完成最小二乘多项式拟合。
- Numpy.reshape(a, newshape, order='C')：这个函数可以用来修改 NumPy 数组的形状。
- Numpy.save(file, arr)：这个函数可以将 NumPy 数组保存到一个 NumPy 的.npy 格式的文件中。
- Numpy.savetxt(fname,X,fmt='%.18e',delimiter='',newline= '\n', header='',footer='', comments='# ')：这个函数可以用来把一个 NumPy 数组保存为一个文本文件。
- Numpy.std(a,axis=None, dtype=None,out=None,ddof=0,keepdims =False)：这个函数可以用来返回给定坐标轴上的坐标值的标准差。
- Numpy.where(condition, [x, y])：这个函数可以根据一个布尔条件从输入的数组中选择数组元素。
- Numpy.zeros(shape, dtype=float, order='C')：这个函数可以用来创建指定形状和数据类型的 NumPy 数组，其元素初始值为 0。

Pandas

下面是常用的 pandas 函数。

- pandas.date_range(start=None,end=None,periods=None, freq='D', tz=None, normalize=False, name=None, closed=None)：这个函数可以用来创建一个固定频率的日期时间索引。
- pandas.isnull(obj)：这个函数可以用来查找 NaN 和 None 值。
- pandas.merge(left, right, how='inner', on=None, left_on=None, right_on=None,left_index=False,right_index=False,sort=False, suffixes=('_x', '_y'), copy=True)：这个函数可以通过类似数据库的列或者索引的联接操作来合并 DataFrame 对象。
- pandas.pivot_table(data, values=None, rows=None, cols=None, aggfunc='mean', fill_value=None, margins=False, dropna=True)：这个函数可以用来创建一个类似 Excel 中使用数据透视表的 pandas DataFrame。
- pandas.read_csv(filepath_or_buffer, sep=',', dialect=None, compression=None,doublequote=True,escapechar=None,quotechar='"', quoting=0, skipinitialspace=False, lineterminator=None, header='infer',index_col=None,names=None,prefix=None,skiprows=None,skipfooter=None,skip_footer=0,na_values=None,na_fvalues=None, true_values=None, false_values=None, delimiter=None, converters=None,dtype=None,usecols=None,engine='c',delim_whitespace=False, as_recarray=False, na_filter=True, compact_ints= False, use_unsigned=False, low_memory=True, buffer_lines=None, warn_bad_lines=True, error_bad_lines=True, keep_default_na= True, thousands=Nment=None, decimal='.', parse_dates=False, keep_date_col=False,dayfirst=False,date_parser=None,memory_map=False,nrows=None,iterator=False,chunksize=None,verbose=False,encoding=None,squeeze=False,mangle_dupe_cols=True,tupleize_cols=False, infer_datetime_format=False)：这个函数可以利用一个 CSV 文件来创建一个 DataFrame。

- pandas.read_excel(io, sheetname, **kwds)：这个函数用作将 Excel 工作表读入一个 DataFrame 中。
- pandas.read_hdf(path_or_buf, key, **kwargs)：这个函数可以从一个 HDF 仓库中返回一个 pandas 对象。
- pandas.read_json(path_or_buf=None, orient=None, typ='frame', dtype=True,convert_axes=True,convert_dates=True,keep_default_dates=True, numpy=False, precise_float=False, date_unit=None)：这个函数可以用一个 JSON 字符串来创建一个 pandas 对象。
- pandas.to_datetime(arg, errors='ignore', dayfirst=False, utc=None, box=True, format=None, coerce=False, unit='ns', infer_datetime_format=False)：这个函数可以用来将一个字符串或者字符串列表转换为日期时间类型（DATETIME）数据。

Scikit-learn

下面是常用的 scikit-learn 函数。

- sklearn.cross_validation.train_test_split(*arrays, **options)：这个函数可以用来将数组拆分为随机的训练数据集和测试数据集。
- sklearn.metrics.accuracy_score(y_true, y_pred, normalize=True, sample_weight =None)：这个函数可以用来返回分类的准确度。
- sklearn.metrics.euclidean_distances (X, Y=None, Y_norm_squared =None, squared =False)：这个函数可以用来计算输入数据的距离矩阵。

SciPy

下面是常用的 SciPy 函数。

scipy.fftpack

- fftshift(x, axes=None)：这个函数用来把零频分量移到频谱中心。
- rfft(x, n=None, axis=-1, overwrite_x=0)：这个函数用来对实数数组

进行离散傅里叶变换。

scipy.signal

- `detrend(data, axis=-1, type='linear', bp=0)`：这个函数用来删除线性趋势或者数据中的常数。
- `medfilt(volume, kernel_size=None)`：这个函数用来对数组进行中值滤波处理。
- `wiener(im, mysize=None, noise=None)`：这个函数用来对一个数组进行 Wiener 滤波处理。

scipy.stats

- `anderson(x, dist='norm')`：这个函数可以对根据指定概率分布生成的数据进行 Anderson-Darling 检验。
- `kruskal(*args)`：这个函数可以用来完成数据的 Kruskal-Wallis H 检验。
- `normaltest(a, axis=0)`：这个函数可以用来测试数据是否符合正态分布。
- `scoreatpercentile(a,per,limit=(),interpolation_method='fraction')`：这个函数用来确定输入的数组中指定百分位数所对应的元素值。
- `shapiro(x, a=None, reta=False)`：这个函数可以通过夏皮罗-威尔克法来检验其正态性。

Python数据分析

（第2版）

数据分析技术能够帮助我们从少量和大量的数据中产生有用的见解。Python拥有许多强大的程序库，已经成为进行各种数据分析和预测建模任务的流行平台。

通过阅读本书，你将学会如何使用Python处理和操作数据，并完成复杂的分析和建模。我们将借助NumPy和Pandas来学习数据的各种操作，例如聚合、连接、附加、清洗和处理缺失值等。本书将介绍如何从各种数据源（例如SQL、NoSQL、CSV文件和HDF5）中存储和检索数据。此外，我们还将学习如何通过可视化库实现数据的可视化，以及各种高级主题（例如信号处理、时间序列、文本数据分析、机器学习和社交媒体分析等）。

本书介绍了众多的Python模块，例如Matplotlib、statsmodels、scikit-learn和NLTK。同时，本书还介绍了Python如何与外部环境（例如R、Fortran、C/C++和Boost库）进行交互。

通过阅读本书，你将学会：

- 在各种平台上安装开源Python模块，例如NumPy、SciPy、Pandas、stasmodels、scikit-learn、theano、keras和tensorflow等；
- 准备和清洗数据，并将其用于探索性分析；
- 使用Pandas处理数据；
- 从RDBMS、NoSQL和分布式文件系统（如HDFS和HDF5）中检索和存储数据；
- 通过开源库（如Matplotlib、bokeh和plotly）实现数据的可视化；
- 了解各种机器学习方法，例如监督学习、无监督学习、概率方法和贝叶斯方法；
- 了解信号处理和时间序列数据分析；
- 掌握图像处理和社交网络分析技能。

异步图书 www.epubit.com

异步社区 www.epubit.com
新浪微博 @人邮异步社区
投稿/反馈邮箱 contact@epubit.com.cn

ISBN 978-7-115-48117-7

美术编辑：董志桢

分类建议：计算机／程序设计／Python
人民邮电出版社网址：www.ptpress.com.cn